Nondeterminism and Language Design in Deep Inference

Dissertation

zur Erlangung des akademischen Grades
Doktor rerum naturalium (Dr. rer. nat.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von
Diplom-Informatiker Ozan Kahramanoğulları
geboren am 18. Oktober 1977 in Ankara, Türkei

Betreuender Hochschullehrer: Prof. Dr. rer. nat. habil. Steffen Hölldobler
Betreuer: Dr. Alessio Guglielmi

Gutachter:
Dr. Alessio Guglielmi, University of Bath
Prof. Dr. rer. nat. habil. Steffen Hölldobler, Technische Universität Dresden
Prof. Dr. Claude Kirchner, LORIA Nancy

Tag der Verteidigung: 21. Dezember 2006

London, April 2007

Acknowledgements

Alessio Guglielmi is the heart and the soul of the development which brought this thesis to being. His thoughts on proof theory and also on every aspect of life influenced me deeply. I am grateful to Alessio for his friendly guidance and his inspiring enthusiasm.

I would like to thank Steffen Hölldobler for accepting the supervision of this thesis. The resourceful and often challenging research environment that he provided at the TU Dresden has contributed a lot to the development and presentation of the ideas here.

Many thanks to Gerhard Brewka for giving me free reign for independent research when I was looking for a topic at the University of Leipzig and for his constant support thereafter. He has also proof-read a preliminary version of this thesis, and provided many helpful suggestions.

I would like to thank Franz Baader, Bernhard Ganter and Michael Schroeder for participating at the examination panel of this thesis.

I am grateful to Claude Kirchner for hosting me at the LORIA in Nancy as a visiting researcher and accepting to referee this thesis. Thanks to members of the Protheo team in Nancy who let me into the family during my stay in 2005, and provided many fruitful discussions that allowed to flourish many interesting ideas. In particular, I would like to thank Pierre-Etienne Moreau and Antoine Reilles for their help with the Tom language.

Many thanks to Steven Eker. His guidance with the Maude language, in particular, his precise and swift answers to my questions by email has been very helpful.

It has been an interesting and exciting experience to evidence the birth of deep inference and the calculus of structures at the TU Dresden. Thanks to Kai Brünnler for introducing me to deep inference and cut elimination at our Sunday-after-jogging-breakfasts when I was a masters student, before I started dreaming about working on proof theory. I have profited from many fruitful discussions with Paola Bruscoli, Roy Dyckhoff, Robert Hein, François Lamarche, Michel Parigot, Charles Stewart, Lutz Straßburger and Alwen Tiu. Their suggestions and comments at various occasions have been very helpful. Robert has also carefully read a preliminary version of this thesis cover to cover and suggested many improvements. Many thanks to Max Schäfer. The graphical proof editor GraPE that Max has developed for the implementations of this thesis is invaluable to me.

This thesis has started in Leipzig, matured in Dresden, and finished in London. I thank Luca Cardelli and Philippa Gardner who evidenced the last phases of this development and supported it with patience.

Without the moral support of friends, this thesis would have never been completed. Many thanks to Vladimir Aleksić, Sören Auer, Sebastian Bader, Sebastian Brandt, Axel Großmann, Hesham Khalil, Carsten Lutz, Maja Miličić, Loïc Royer,

Barış Sertkaya, Olga Skvortsova and Anni-Yasmin Turhan. Thank you all for sharing a laugh, beer, or knowledge.

I would like to thank the members of the Gingapura capoeira group and other friends in Dresden, Leipzig and London. Many thanks to my friends in Turkey who welcomed me, as if I have never left, at my rare visits. Thanks to my family!

The greatest thanks is to Sheila Eilenberg who made me understand and appreciate the meaning of the word 'angel' truly. *Thank you Sheila!*

This work has been written with the financial support of the DFG Graduiertenkolleg 446 at the University of Leipzig.

Abstract

This thesis studies the design of deep-inference deductive systems. In the systems with deep inference, in contrast to traditional proof-theoretic systems, inference rules can be applied at any depth inside logical expressions. Deep applicability of inference rules provides a rich combinatorial analysis of proofs. Deep inference also makes it possible to design deductive systems that are tailored for computer science applications and otherwise provably not expressible.

By applying the inference rules deeply, logical expressions can be manipulated starting from their sub-expressions. This way, we can simulate analytic proofs in traditional deductive formalisms. Furthermore, we can also construct much shorter analytic proofs than in these other formalisms. However, deep applicability of inference rules causes much greater nondeterminism in proof construction.

This thesis attacks the problem of dealing with nondeterminism in proof search while preserving the shorter proofs that are available thanks to deep inference. By redesigning the deep inference deductive systems, some redundant applications of the inference rules are prevented. By introducing a new technique which reduces nondeterminism, it becomes possible to obtain a more immediate access to shorter proofs, without breaking certain proof theoretical properties such as cut-elimination. Different implementations presented in this thesis allow to perform experiments on the techniques that we developed and observe the performance improvements. Within a computation-as-proof-search perspective, we use deep-inference deductive systems to develop a common proof-theoretic language to the two fields of planning and concurrency.

Contents

List of Figures

CHAPTER 1

Introduction

An important part of the research effort in theoretical computer science is focused on providing a mathematical foundation to formal languages like specification and programming languages. Proof theory, which was originally set up as an area of mathematics that studies the concepts of mathematical proof and provability, provides powerful tools for a rigorous formal treatment of formal languages.

Although semantics plays a crucial role in the development of proof theory, the main concern of proof theory is the formal syntax of logical formulae and syntactic presentations of proofs. Therefore proof theory can be regarded as logic from the syntactic point of view.[1] An important topic of research in proof theory is the relation between finite and infinite objects. In other words, proof theory investigates how infinite mathematical objects are denoted by finite syntactic constructions, and how facts concerning infinite structures are proved by finite proofs. This is particularly important for computer science, studying computers as syntactic engines, which perform syntax manipulations for performing computations by using finite resources, i.e., memory and time. Especially from the point of view of formal theory of language, which is more concerned with the connectives of a logical system and their relations, proof theory provides the appropriate mathematical techniques and tools for a formal analysis of computer languages. By restricting itself to finitary methods, proof theory studies the objects that computers can deal with, which are per se finite [**Str03a**].

From the point of view of computer science, perhaps the most influential work on proof theory, around which major developments took place, is Gentzen's sequent calculus [**Gen34, Gen35, Gen69**]. In the sequent calculus, inference rules of a deductive system directly model syntactical properties of logical connectives. This way, they provide a finite description of an infinite set of formulae which are valid. This provides a purely syntactic view of logic, the reliability of which is assured by the cut elimination property.

The cut elimination property, which is central to proof theory, provides a formal measure of rigour of the proof theoretical systems. A proof theoretical system has the cut elimination property if for every proof in the system that uses the *cut rule*, there is a proof with the same conclusion that does not use the cut rule. Although the cut rule varies from a proof theoretical system to another, it always expresses the transitivity of the logical consequence relation. The usage of the cut rule demonstrates the usage of a lemma in the proof. Thus, the cut elimination property states that if a logical expression can be proved by using lemmas, it can be proved also without using any lemma. The cut elimination property has

[1] In contrast, model theory studies logic with an emphasis on semantics.

implications such as consistency and completeness of the system being addressed (see, e.g., [**Gen35, Brü03b, Str03a**]).

In contrast to the sequent calculus view of logic, logic in the tradition of Hilbert and Tarski was primarily semantics oriented. The central interest was in model theory and problems were mainly inspired by set theory. In general the emphasis was on infinite mathematical structures. However, computer science is particularly interested in finite structures, and the formal theory of language is more concerned about the connectives of a logical system, and their relations, than in traditional models. Furthermore, syntax in the sequent calculus (and in its off-springs) is much closer to operational semantics, which describes how programs are interpreted as sequences of computational steps. In comparison to methods based on traditional semantics, this is also a clear advantage of proof theory with respect to applications.

1.1. Declarative Programming and Proof Theory

Standing at the core of theoretical computer science and being concerned with the relation between intuitive proofs and formal systems, proof theory provides theoretical foundations for declarative programming. In contrast to imperative programming, in declarative programming the intention is to describe *what* the user wants to achieve, instead of providing instructions which describe *how* the machine is going to achieve it. In such a perspective, it is crucial for the computation of the machine to meet the intuition of the user. Proof theory provides the theoretical foundations for the two declarative programming paradigms of functional programming and logic programming. While the theoretical foundations of the functional programming paradigm are given by the proof theoretical concept of proof normalization (or proof reduction), logic programming is brought to theoretical grounds by means of proof search (or proof construction) in deductive systems.

1.1.1. Functional Programming and Proof Theory. The relation between the functional programming paradigm and proof theory is established by the *Curry-Howard isomorphism* or the formulae-as-types correspondence (see, e.g., [**How80, SU99**]): Curry-Howard isomorphism describes a correspondence between deductive systems, as they are studied in proof theory, and computational systems, as they are studied in type theory. More precisely, a formula corresponds to a type and a proof of that formula to a term of the corresponding type. For example, natural deduction proofs of intuitionistic logic correspond to terms of the simply typed λ-calculus. This mapping between proofs and terms is an isomorphism because a normalization (cut elimination) step of the proof in the logical system corresponds exactly to a normalization step of the λ-term, which in turn is a computation step in functional programming.

Such an interpretation of proofs is powerful enough to capture many aspects of computation, including concurrent computations (see, e.g., [**Abr93**]). However, a large and growing number of applications do not fit well in this paradigm [**And01**]. This is because functional programming is concerned with programs that are meant to end and return a result. The cut-elimination procedure (and the strong normalization theorem) give a convenient abstraction of what is going on in the execution of such programs, but many pieces of software do not fall into that category. In [**And01**], Andreoli lists the following applications as examples of such programs: An operating system; an air-traffic control system which ensures that plane routes do not collide; an electronic commerce broker, whose role is to mediate between a

set of service and good providers, and a set of customers; or a web browser which organizes the interaction between a client and a set of servers on the Internet. The main characteristic of all such applications is that instead of taking some input and returning a result, they are concerned with the coordination of entities which are external to them, and with which they have to continuously interact. Andreoli makes the following further remark:

> "The point is here not to say that the intuitions behind the functional programming paradigm are totally inadequate for the class of applications mentioned above. Obviously, an operating system or an electronic commerce broker will need, at some points in their execution, to launch a process to perform a functional computation that takes input, executes at some time as a black-box and produces output. But the functional programming paradigm does not capture the overall picture of the behavior of the application. It ignores a number of essential characteristics of these applications, in particular true-nondeterminism, or the manipulation of the partial information, not to mention a huge set of deeply woven issues such as security, robustness, etc. Robustness, for instance, is not just a nice feature to have in an application such as an air-traffic control system, it is an absolute requirement, almost the "raison-d'être" of the application, and a programming paradigm which would try to capture the computational essence of this application without taking into account this aspect is deemed to fail."

The logic programming paradigm, which is central to this thesis, addresses the issues related to above mentioned applications in a more satisfactory manner, while remaining on formal grounds.

1.1.2. Logic Programming and Proof Theory. Logic programming can be given a foundation in the sequent calculus by viewing computation as the process of building a cut-free proof bottom-up: A logic program is a conjunction of formulae. The input to the program is another formula, called the goal. The computation is the search for a proof (also called proof construction) showing that the goal is a logical consequence of the program [**Str03a**]. Thus, in the logic programming paradigm (or paradigm of proof search as computation) searching for a proof corresponds to the execution of a logic program and a proof corresponds to the trace of a successful execution. From the point of view of imperative programming, a logic program can be considered also as follows: The formulae identify instructions of a program whereas (incomplete) proofs identify states. Each instruction states a set of possible state transitions of the program.

Historically, logic programming can be traced back to the programming language Prolog [**Llo87**], which is based on the first-order classical theory of Horn clauses.[2] Prolog was initially introduced as an application of the SLD-resolution[3]

[2][**Rob00**] is a brief survey on the evolution of declarative programming including the development of Prolog, starting from the early days of modern logic.

[3]SLD-resolution is resolution with a selection function for Horn caluses. A Horn clause has exactly one positive literal. In SLD-resolution, the only positive literal of a Horn clause is selected to be resolved upon, i.e., replaced in the goal clause by the conjunction of negative literals which form the body of the clause.

method. However, the SLD-resolution perspective turned out to be difficult to extend the pure language of Horn clauses to more expressive languages, without sacrificing logical purity. In particular, concepts like modular programming or abstract data types, which are common in modern programming languages, cannot be considered in a resolution setting.

In order to overcome the short-comings of Prolog with respect to above mentioned points, several approaches have been considered [**MNPS91**]: One is mixing the concepts of other programming languages into Horn clauses, or extending an interpreter by certain non-logical primitives that provide aspects of the missing features. "assert" and "retract" commands, and the "cut" primitive of popular Prolog interpreters are examples for this. Another approach is resorting to more expressive logics, which capture the desired previously missing mechanism.

Although the former approaches lead, in general, to immediate and efficient extension of the language, they imply a depart from mathematical rigor. For instance, previously available logical semantics and declarative reading of the programs become hampered by using the non-logical constructs of the extended language. The latter approach, on the other hand, brings about the question of which logic should be employed, such that efficient implementations can still be possible. The solution can be found somewhere between the two extremes of Horn logic, which is weak but proof search can be implemented efficiently, and more expressive logics, for which all purpose theorem provers have to serve as interpreters. This also brings about problems attached to the efficiency of the proof search implementations of more and more expressive logics.

The first account of logic programming, following this latter approach, was given in [**MN86, NM90**], where Miller and Nadathur used the sequent calculus to examine design and correctness issues for logic programming. The notion of *uniform provability*, introduced in [**MNPS91**], provides a criterion in these lines, for judging whether a given logical system is an adequate basis for a logic programming language.

A *uniform proof* is a proof that can be found by a goal-directed search, i.e., the logical connectives in the goal can be interpreted as search instructions. In other words, when sequents are single-conclusion, a uniform proof is a cut-free proof in which every sequent with a non-atomic right-hand side is the conclusion of a right introduction rule. An interpreter attempting to find a uniform proof of a sequent would directly reflect the logical structure of the right-hand side (the goal) into the proof being constructed. In a uniform proof, left introduction rules are used only when the goal formula is atomic, and as part of the *backchaining* phase, in which the meaning of an atomic formula with respect to the program is extracted from the program clauses. This is analogous to applying a Horn clause to an atomic query in Prolog.

A specific notion of goal formula and program clause along with a proof system is called an abstract logic programming language [**BG03**] if a sequent has a proof if and only if it has a uniform proof. First order and higher order variants of Horn clauses paired with classical provability [**NM90**] and hereditary Harrop formulae[4] paired with intuitionistic provability [**MNPS91**] are two examples of abstract logic programming languages.

[4]Hereditary Harrop formulae is a generalization of Horn clauses.

The above mentioned ideas resulted in the development of λ-Prolog [**Mil95**] which is based on higher-order hereditary Harrop formulae [**MNPS91**]. Thus, λ-Prolog supports modular programming, abstract data types and higher order programming. Linear logic refinement of λ-Prolog resulted in the programming languages Lolli [**Hod94**] and LO [**AP91**]. Lolli provides various forms of abstraction (modules, abstract data types, and higher order programming), but lacks primitives for concurrency. LO, on the other hand, provides some primitives for concurrency, but lacks abstraction mechanisms.[5]

Abstract logic programming languages make it possible to consider the expressive logics within the realm of programming. However, these languages usually impose syntactic restrictions on the formulae, as in the Horn clauses and hereditary Harrop formulae: Typically, as in uniform proofs, sequent $\Delta \vdash G$ represents the state of an idealized logic programming interpreter in which the logic program is Δ and the goal is G. These two classes of formulae are duals of each other in the sense that a negative subformula of a program clause is a goal formula. Goal formulae are processed immediately by a sequence of invertible right rules and program clauses are used via a focused application of left-rules, i.e., backchaining. However, this view of the logic programs focuses attention only on fragments of logical systems for a computational interpretation. In other words, such restrictions, in general, do not allow to use these logics directly in their full expressive power in a logic programming setting.

Andreoli's focusing proofs [**And92, And01**] attacks the problem of bringing linear logic to more efficient grounds in proof construction without imposing any restrictions on the syntax of the formulae. Although proof construction is, by nature, a highly nondeterministic process, not all the nondeterminism in proof search is meaningful. Making this observation, Andreoli analyzed the structure of proof search in linear logic and classified the logical connectives into two sets of connectives, namely asynchronous (deterministic) and synchronous (nondeterministic) connectives, which are de Morgan duals of each other: Asynchronous connectives are those whose right-introduction rule is invertible and synchronous connectives are those whose right-introduction rule is not invertible; that is, the success of applying a right-introduction rule for a synchronous connective requires information from the context. A formula is asynchronous or synchronous depending on the top level (main) connective of the formula.

Given these distinctions, Andreoli showed that a complete bottom-up proof search procedure for cut-free proofs in linear logic can be described roughly as follows: First decompose all asynchronous formulae and when none remains, pick some synchronous formula, introduce its top-level connective and then continue decomposing all asynchronous subformulae that might arise. Thus interleaving between asynchronous and synchronous reduction yields a highly normalized proof search mechanism. Proofs built in this fashion are called focused proofs.

As a consequence of the completeness of focused proofs for linear logic, linear logic can be seen as a logic programming language that captures the notion of uniform proofs and backchaining. The language "Forum" [**Mil96**], which exploits

[5] [**Mil04**] is an overview of the proof search paradigm, focusing on logic programming based on linear logic.

these ideas, is a logic programming specification of all of linear logic. Forum modularly extends λ-Prolog, Lolli, and LO. Forum allows specifications to incorporate both abstractions and concurrency.

The following are further examples of other linear logic programming languages: ACL [**KY93a**], by Kobayashi and Yonezawa, is an asynchronous calculus in which the send and read primitives were essentially identified by two complementary linear logic connectives. Lincoln and Saraswat developed [**LS92**] a linear version of concurrent constraint programming and used linear logic connectives to extend previous languages in this paradigm.

1.1.3. Non-commutativity. Proof theoretical insights on classical, intuitionistic and linear logic have found successful applications in many areas of computer science.[6] Linear logic, also due to its resource sensitive features, is widely recognized as a logic of concurrency (see, e.g., [**DQ03, Mil92, EW94**]). The proof theory underlying it faithfully represents some aspects of concurrent computation. Further, linear logic is also well suited for modeling concepts of action and change as they appear in planning problems (see Section 8.2). However, one important limitation of linear logic (and also others) from the point of view of computer science, especially with respect to these application areas, is its inability of dealing with non-commutativity. Sequential composition (of actions, processes, programs, etc.), which is naturally expressed by non-commutative operators, is not specifiable in traditional logics without resorting to terms of the logic. For instance, where a and b are two actions, and $\lhd$ is a non-commutative operator, the syntax $a \lhd b$ can denote the plan where first a and then b is executed. However, due to the lack of non-commutative operators this is usually achieved by encoding such a structure into the terms of the language, e.g., $Do(b,\, Do(a, S))$.

Given that the logic at hand is complexity-wise expressive enough, capturing the structure of the application domain by means of function symbols is something which can always be done. For instance, given that first-order Horn logic is Turing-complete, this logic would suffice for any potential applications of logic programming, simply by expressing everything at the term level. However, it is much more desirable to have the structural content of programs and computations reflected into the connectives of the logic. This way, one can use logic in a non-trivial way, e.g., to do reasoning and draw interesting conclusions about the application domain, but not as an elegant interface between the application domain and the user.

Because of its importance in computer science applications, non-commutativity has been studied by various authors in the context of proof theory: Lambek calculus [**Lam58**], which aimed at modeling syntax of natural language, was the first logic studying non-commutativity. After the introduction of linear logic in 1987, different approaches for non-commutative logics have been studied in the lines of linear logic: By introducing restrictions on the exchange rule, which is the rule responsible for commutativity in the sequent calculus, the first approaches resulted in different versions of purely non-commutative logics (Yetter's cyclic linear logic [**Yet90**], Abrusci's non-commutative logic with two negations [**Abr91**], non-commutative logic in [**LMSS90**]): In these logics, there are only non-commutative multiplicative operators. However, many applications in computer science require commutative

[6] [**Ale94**] is a survey on applications of linear logic to computer science. Although it is some what out of date, it gives a good overall picture of early development around linear logic.

and non-commutative operators at the same level. For instance, in concurrency theory parallel and sequential composition of processes are equally important, thus they need to be represented at the same level. Furthermore, in process algebras, e.g., CCS [**Mil89**], usually the non-commutative prefix operator is self-dual.

Another approach which attacks this problem is Abrusci and Ruet's non-commutative logic [**AR00, Rue00**]. This logic admits two pairs of multiplicative connectives, one commutative and one non-commutative. The non-commutative conjunction and disjunction are duals of each other, thus this logic does not admit a self-dual non-commutative operator.

Retoré's *pomset logic*, introduced in [**Ret93**], fulfills these requirements: *Pomset logic* is an extension of multiplicative linear logic with a self-dual non-commutative operator which is intermediate between multiplicative conjunction and multiplicative disjunction. Thus, this self-dual non-commutative operator resembles the sequential composition in process algebras. However pomset logic lacks a sequent calculus system with cut-elimination property. Guglielmi's system BV is a logic which is very similar to pomset logic. In fact, Guglielmi [**Gug07**] and Straßburger [**Str03a**] conjecture that these logics are equivalent.

System BV cannot be designed in a standard sequent calculus, as Tiu showed in [**Tiu01, Tiu06b**]. This system is designed in the *calculus of structures*, a proof theoretical formalism with *deep inference*. The idea of deep inference delivered systems with interesting and exciting properties for existing logics and brought new insights to proof theory of these logics: In his PhD thesis, [**Brü03b**] Brünnler studies classical logic in the calculus of structures; Straßburger's PhD thesis [**Str03a**] presents systems for different fragments of linear logic. In [**SS05**] and [**HS05**], Stewart and Stouppa, and Hein and Stewart, respectively, give systems for a collection of modal logics. In [**Tiu06a**], Tiu presents a local system for intuitionistic logic. *One of the topics, which I discuss in this thesis, is implementations of the systems with deep inference.* The implementations that I present in this thesis are the first implementations of the deep inference systems. Apart from the academic interest in the implementations of the deep inference systems, the implementations of the systems that are designable only in the presence of deep inference, e.g., system BV, should be of interest for the potential applications of these systems.

1.2. Proof Theory with Deep Inference

Developing new representations of logics, which address properties that are central to computer science applications, has been one of the challenging goals of proof theory. In this regard, a proof theoretical formalism must be able to provide a rich combinatorial analysis of proofs while being able to address issues which are important for computer science applications. The calculus of structures [**Gug07**], introduced in 1999 by Guglielmi, is a proof theoretical formalism with such a perspective. Like in other proof theoretical formalisms, e.g., natural deduction, the sequent calculus, or proof nets [**Gir87**], in this formalism logical systems are specified. However, the calculus of structures is motivated by computation, and thus it is well suited for dealing with aspects of computation such as *locality*, *modularity* and *non-commutativity*.

1.2.1. Deep Inference. The calculus of structures is a generalization of the sequent calculus. Structures are expressions intermediate between formulae and sequents which unify these two latter entities, i.e., the calculus of structures replaces

the notions of sequent and formulae of the sequent calculus with the notion of structure. The main feature that distinguishes this formalism is *deep inference*: In contrast to the sequent calculus, the calculus of structures does not rely on the notion of main connective, and permits the application of the inference rules at any depth inside a structure. Thus, sequent calculus systems can be freely designed in the calculus of structures, where the inference rules are applied only at the top level. However, it becomes possible to design other systems, which allow for more freedom in the application of the inference rules. This provides a combinatorial richness where inference rules can be applied in many more ways.

The deep inference feature does not only provide a rich combinatorial analysis of the logic being studied, but also brings shorter proofs than any other formalism supporting analytical proofs [**Gug04c**]: Applicability of the inference rules at any depth inside a structure makes it possible to start the construction of a proof by manipulating and annihilating substructures.

In order to see this on an example consider the following two proofs of a classical logic formula, respectively, in the one-sided sequent calculus system GS1p, *Gentzen-Schütte* system [**TS96**] and system KSg of the calculus of structures [**Brü03b**] (see Section 2.4). In the proofs, shaded area indicate the places where the inference rules are applied.

$$\dfrac{\mathsf{R}\wedge\dfrac{\mathsf{Ax}\dfrac{}{\vdash a\ ,\ \bar{a}}\qquad \mathsf{R}\vee\dfrac{\mathsf{Ax}\dfrac{}{\vdash b\ ,\ \bar{b}}}{\vdash b\vee\bar{b}}}{\vdash a\ ,\ \bar{a}\wedge(b\vee\bar{b})}}{\mathsf{R}\vee\dfrac{}{\vdash a\vee(\bar{a}\ \wedge\ (b\vee\bar{b}))}} \qquad\qquad \mathsf{ai}{\downarrow}\dfrac{\mathsf{tt}{\downarrow}\dfrac{}{\vdash\top}}{\mathsf{ai}{\downarrow}\dfrac{}{\vdash a\vee(\bar{a}\wedge(b\vee\bar{b}))}}{\vdash a\vee(\bar{a}\wedge(b\vee\bar{b}))}$$

Although the sequent calculus system GS1p and the calculus of structures system KSg appear to be very similar, the inference rules in the former system can be applied only at the main connective whereas the inference rules of the latter can be applied at any depth inside a structure, and their application this way results in shorter proofs.

The word *structure* is used in philosophical logic to indicate a certain kind of expression used in formalisms where the emphasis is on the structural component of deduction. The calculus of structures is a deductive formalism, where the deduction is performed directly on structures, instead of mixed expressions involving sequents, structures and formulae, as for example in the display calculus [**Bel82**], another formalism with some kind of deep inference.

Another important property of the calculus of structures is that derivations are not trees like in the sequent calculus, but chains of inferences: In the sequent calculus, due to the multiple premise inference rules, the derivations branch while going up. The information between two branches in a proof is the meta-level information, in contrast to object-level information captured by the language of the logic. The co-existence of meta-level and object-level in the sequent calculus systems does not have any side-effect for classical logic, because the branching in a derivation corresponds to classical conjunction. However, in some cases, like linear logic this branching causes a mismatch between the logical operators of the logic and the meta level of the proof theoretical system [**Gug03**]. This mismatch has

proof theoretical consequences. Furthermore, in an implementation the branching due to the meta-level requires an additional machinery for the representation of the meta-level information (see Chapter 3). Because the information about the derivation, which is represented at the meta-level of the sequent calculus systems, is represented at the object-level of systems of the calculus of structures, there are no multiple premise inference rules. Thus, derivations are chains of inferences, rather than trees. In order to see this on an example consider the proof below, which is the calculus of structures analogue of the sequent calculus proof on the left-hand side above.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \top}\,\mathsf{tt}\downarrow}{\vdash a \vee \bar{a}}\,\mathsf{ai}\downarrow}{\vdash (a \vee \bar{a}) \wedge (b \vee \bar{b})}\,\mathsf{ai}\downarrow}{\vdash a \vee (\bar{a} \wedge (b \vee \bar{b}))}\,\mathsf{s}$$

The freedom provided by deep inference makes it possible to design deductive systems which are otherwise impossible. However, this great power, as usual, comes along with a responsibility attached to it. Guglielmi makes the following remark in [**Gug07**]:

> "The freedom allowed by this formalism (the calculus of structures) is dangerous. One could use it in a wild way, and lose every proof theoretical property."

Thus, it is important to understand and define the methodologies necessary for being able to benefit from the use of the new possibilities provided by deep inference. The reasons for this extra care is that deep inference challenges the previously known proof theoretical techniques. For instance, cut elimination in systems with deep inference is completely different from that in the sequent calculus: Because of the absence of a notion of main connective in the systems with deep inference, the cut elimination technique of the sequent calculus, which depends on the existence of main connective, cannot be carried over to the calculus of structures. Because of this, new techniques were developed, i.e., Guglielmi's *splitting* technique [**Gug07**] (see Chapter 5) and *decomposition* (see, e.g., [**Str03a**]). These techniques exploit the fact that the cut rule in the calculus of structures can be made atomic, and atomic cuts are much simpler objects than generic cuts (see, e.g., [**Brü03a**]).

1.2.2. Modularity. The absence of meta-level in the calculus of structures allows to observe a top down symmetry in the inference rules. This symmetry is natural, when an inference rule is seen as an implication such that the premise of the rule implies the conclusion: The *contrapositive* of an inference rule delivers another inference rule which is sound. That is, in the calculus of structures, by flipping a rule up-side down and negating everything, it is possible to obtain a dual rule. In each system, this results in two groups of inference rules: The down fragment which is complete, and an up fragment. The down rules are denoted by $\downarrow$ whereas up rules are denoted by $\uparrow$. This duality is perhaps most spectacular when the atomic interaction rule (the analogue of the axiom in the sequent calculus) and the cut rule are observed: The duality of these rules, which is immediate in proof nets, become observable in the inference rules of the systems of the calculus of

structures. For instance, for classical logic the atomic interaction rule and the cut rule are, respectively, of the following form (S denotes the context in which the rule is applied, $\mathtt{tt}$ and $\mathtt{ff}$ are the constants for true and false, respectively):

$$\mathsf{ai}{\downarrow}\,\frac{S(\,\mathtt{tt}\,)}{S(\,a\,\vee\,\bar{a}\,)} \qquad\qquad \mathsf{ai}{\uparrow}\,\frac{S(\,a\,\wedge\,\bar{a}\,)}{S(\,\mathtt{ff}\,)}$$

The availability of this duality and the decomposition theorems in [**Brü03b, Str03a**] bring about a modular behavior which is important for computer science applications: Because of the availability of the dual rules of a system, many equivalent systems can be obtained just by adding and removing up-rules to a system. For instance, for a system with 3 up rules, 8 equivalent systems can be obtained: The powerset of the set of these 3 rules contains 8 sets, thus each of these 8 sets of rules can be safely combined with the down fragment, resulting in equivalent systems. This way, a system can be easily tailored according to the needs of the application domain. Further, the decomposition results presented in [**Brü03b, Str03a**] point out the property of being able to separate any given derivation or proof into several phases, each of which consists of applications of rules coming from mutually disjoint fragments of a given logical system.

1.2.3. Locality. In the calculus of structures it is possible to design local systems. A system is local if every inference rule of the system is local in the sense that at each application of an inference rule a bounded amount of information about the logical expression is involved. This is important for computer science applications because the application of a local rule consumes only a bounded amount of computational resources, i.e., memory and time. In the sequent calculus, for instance, many deductive systems contain inference rules that are not local. The *contraction* rule (RC) of the classical logic, and the *with* ($\&$) and *promotion* ($!$) rules of linear logic are examples to such rules:

$$\frac{\vdash \Phi, A, A}{\vdash \Phi, A}\,\mathsf{RC} \qquad \frac{\vdash A, \Phi \quad \vdash B, \Phi}{\vdash A \,\&\, B, \Phi}\,\& \qquad \frac{\vdash A, ?B_1, \ldots, ?B_n}{\vdash\, !A, ?B_1, \ldots, ?B_n}\,!$$

When the contraction rule is applied bottom-up, the formula A has to be duplicated. However, there is no bound on the size of this formula. A similar situation occurs when the rule $\&$ is applied bottom up: The context Φ of the formula $A\&B$ has to be copied, however there is no bound on Φ. The situation with the promotion rule is similar: In a bottom-up application of this rule, every formula in the context has to be checked to have the form $?B$. The amount of computational resources necessary for an instance of such non-local rules cannot be determined before-hand, in contrast to local rules. For example, the rule $\mathsf{R}\wedge$ is a local rule because an application of this rule involves replacement of connectives (pointers) rather than copying of formula of unbounded size.

$$\frac{\vdash \Phi,\, A \qquad \vdash \Psi,\, B}{\vdash \Phi,\, \Psi,\, A \wedge B}\,\mathsf{R}\wedge$$

Local systems are obtained by replacing each non-local inference rule with equivalent local rules. For instance, a local system for classical logic is obtained by replacing the non-local generic contraction rule with the atomic contraction rule, the computational resources required by which is bounded by the size of an

atom. Locality is respected in the systems for classical logic, intuitionistic logic and linear logic, presented in [**Brü03a**], [**SS05**], [**Tiu06a**], and [**Str02**], respectively. Furthermore, the systems for non-commutative logics presented in [**Gug07, Gb01**] are also local.

1.2.4. Non-commutativity in the Calculus of Structures. The calculus of structures was originally conceived in order to introduce system BV which extends multiplicative linear logic with a self-dual non-commutative operator. As Tiu showed in [**Tiu01, Tiu06b**], deep inference is crucial for designing system BV because any restriction on the depth of the inference rules of system BV would result in a strictly less expressive logical system.[7] Due to the self-dual non-commutative logical operator, system BV is of particular interest for applications where sequentiality plays an important role. In particular, as Bruscoli showed in [**Bru02**], the non-commutative operator of BV captures precisely the sequentiality notion of process algebra, e.g., CCS.

In [**GS02**], Guglielmi and Straßburger introduced a system, called NEL, which extends system BV with the exponentials of linear logic. System NEL is a conservative extension of system BV. Although it is unknown whether multiplicative exponential linear logic is decidable or not, in [**Str03c, Str03a**], Straßburger showed that system NEL is undecidable. *In Chapter 6, I show that the decision problem in system* BV *is NP-complete.*

In this thesis, I present implementations of the systems of the calculus of structures, and study the issues, in these systems, related to nondeterminism in proof search. The non-commutative operator of systems BV and NEL allow to express sequential composition of computational entities such as plans, processes, programs, etc. Further, the parallel composition of such entities can be naturally mapped to the multiplicative disjunction of these logics. By exploiting this observation and the expressive power due to Turing-completeness of system NEL, in a logic programming setting, I will give the foundations of a common logical language for the two fields of planning and concurrency. This language should be helpful to bring these two fields closer so that techniques can be exchanged.

1.3. Planning and Concurrency

Reasoning about action and change, as a branch of artificial intelligence, has been one of the main-stream fields of computer science since logic had been considered in [**McC59**] as a tool for simulating *commonsense behavior* by computers. There classical logic was proposed to represent facts about the consequences of actions in order to perform reasoning on these actions. While this approach has evolved to what was later called the situation calculus [**MH69**], many approaches and debates followed these ideas, addressing different aspects and problems related to *actions and causality* (see Section 8.1). Planning, which is motivated by methods of reasoning about action, focuses on automated exploration of the state space of these actions: Based on the assumption that all the necessary information about the world is easily obtained, an agent can use the percepts provided by the environment to build a complete and correct model of the current world state. Then,

[7] If the conjecture on the equality of pomset logic and system BV is true, this result explains why no sequent calculus system with cut-elimination property for pomset logic could be designed so far.

given a goal, it can call a suitable planning algorithm to generate a plan of action, provided such a plan *exists* [**RN02**].

Planning is a quite developed field which is very much motivated by logic, that is, logic and declarative methods have been central in planning. The main developments in this field are motivated by efficiency concerns: As a result of the research effort in this field faster and faster planners, which employ highly optimized heuristic methods, are being developed (see Section 8.1). However, inspired by empiric methods, the theoretical insights provided by these approaches usually relate to the efficient exploration of the search space rather than the true nature of these problems. As a consequence, while these approaches provide satisfactory solutions for domains of limited size on which they are tested, they can be rarely employed in real-world planning domains.

Concurrency theory is another popular field of computer science which has evolved independently. By means of process algebras [**BPS01**] concurrency theory focuses on universally quantified queries on systems where communicating, concurrent processes change the system state. Typical examples for such queries are deadlock freeness of a system or verification of some security protocol. Such tasks require a rich arsenal of formal methods, for instance, for proving that two processes are equivalent up to their "behavior". Concurrency theory studies techniques which provide a global view of a concurrent system, so that a structural analysis of such systems becomes possible.

The fields of planning and concurrency have evolved independently because they aim at solving tasks that are different in perspective. However, the problems addressed in these two fields are similar in nature. When processes are viewed as actions or plans, and vice versa, the difference between these two fields can be seen as the different quantification of the queries of the problems being addressed: Planning formalisms focus on finding a plan that solves a given planning problem, by means of *existentially* quantified queries: A typical question in planning is "does there exist a plan which brings the agent from the initial state to the desired goal state?". On the other hand, the focus in concurrency theory is on *universally* quantified queries, e.g., deadlock freeness, which imposes a global view of all the executions of the system being examined.

1.3.1. A Common Language for Planning and Concurrency. In concurrency theory, the universal quantification on the queries imposes a global view of the concurrent systems being studied. When carried to planning, such a view has the potential to provide the theoretical insight for a deeper understanding of the problems being attacked in planning. By studying the specification of a planning problem, one can observe the global behavior of such a specification, similar to the specification of a concurrent system, and, for instance, compare different plans solving the problem. Let us consider a simple planning scenario which is helpful to demonstrate these ideas: In this scenario there are two tables. On Table 1 there are four blocks which are stacked on top of each other as shown on the left-hand side of the Figure 1.1. The only available action takes a block from Table 1 and puts it on Table 2. The goal of the problem is moving three of the blocks from Table 1 to Table 2. Because block a is stacked on blocks c and d, blocks c and d cannot be moved before block a. Similarly, block d cannot be moved before block b. There are five different sequences of actions which solve this planning problem, namely the following plans ($\lhd$ denote the sequential composition of actions, and a

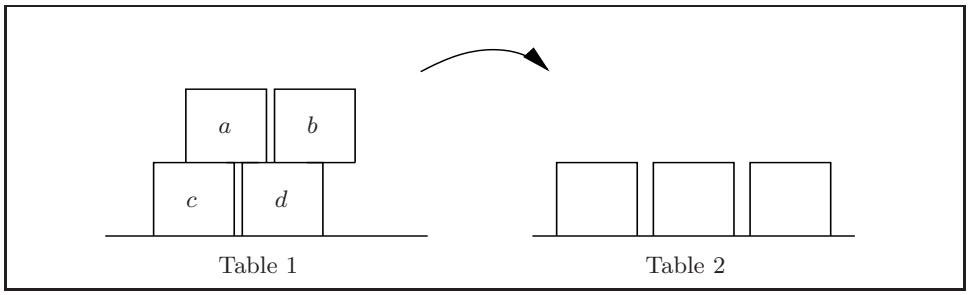
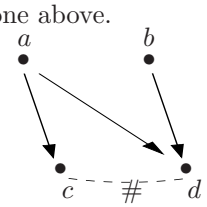


FIGURE 1.1. A simple planning problem

the action of moving block a from Table 1 to Table 2, similarly for blocks b, c and d):

$$a \lhd c \lhd b \qquad b \lhd a \lhd c \qquad a \lhd b \lhd c \qquad a \lhd b \lhd d \qquad b \lhd a \lhd d$$

Each one of these plans is a typical output of a planning algorithm. Although they all bring the agent to the desired goal state, the relationship between these five different plans is difficult to observe. In contrast to such a view of this planning problem, let us consider a different view, represented by the graph below. In this graph # is read as a conflict relation in the sense that nodes connected with this relation cannot co-occur in a plan. The partial order of the nodes determine in which order these nodes can-occur, that is, a node appearing below another one must occur strictly after the one above.



This graphical representation allows to observe all the actions which can be executed in the planning scenario above. All of the five plans above can be easily read from this graph. In fact this is a graphical representation of a model for concurrency, namely a *labelled event structure* [**SNW96, WN95**]. Labelled event structures (LES) is a behavioral model of concurrency. In a LES the independence and causality between events is expressed as a partial order, and the nondeterminism is expressed by a conflict relation. *In this thesis, I argue that a logical common language for planning and concurrency, with such a semantics can serve as a bridge for importing formal techniques from concurrency to planning, for instance, for establishing a notion of plan equivalence.*

A common language for planning and concurrency is also meaningful from the point of view of concurrency theory. Because of the duality between existential and universal quantification, concurrency theoretical queries can be treated as planning problems. For instance, verification of security protocols is a problem domain which is commonly addressed in concurrency theory and can be easily put as a planning problem: "Is there a sequence of actions that an intruder can undertake so that

he can break a security protocol?". The Dolev-Yao model of security protocols [**DY83**] is well suited for such a representation.

1.3.2. Causality, Independence, Nondeterminism. In general, concurrency theory deals with analyzing properties of processes in a system where the processes interact with each other. This interaction can be in the form of a process consuming the resources which are produced by another process or the synchronization of two process that are running in parallel via a *hand shake* operation. These two sorts of interaction brings parallel and sequential composition into focus. In a process algebra parallel and sequential composition are at the same level because they are equivalently important notions for expressing concurrent processes. However, in planning the emphasis is on sequential composition. The partial order planners and graph planners, which mainly focus on improving the performance of planners, do not provide a satisfactory treatment of parallel behavior within the borders of logic, because they cannot handle resource conflicts between competing actions. On the other hand, the treatment of concurrency within the approaches to reasoning about action, where concurrency is defined over the parametrized time spans shared by the actions, does not capture the independence and nondeterminism inherent in the system (see Section 8.1).

In fact, finding the right logic with which to specify and reason about plans and to get a satisfactory semantic treatment of concurrent actions simultaneously is a challenging task. One of the obstacles to this task is the *frame problem* [**MH69**]. Informally, the frame problem occurs when the formal language expresses what changes, but does not express what stays the same (see Section 8.1). The underlying logic must be powerful enough to express causality in a simple way without raising the frame problem. Further, an explicit treatment of resources is crucial in order to express the independence and nondeterminism in the system. Another ingredient is being able to express parallel and sequential composition of actions at the same level. These conditions must be fulfilled without resorting to function symbols, so that the structure of the problem can be captured at the logical level, rather than term level, so that logic can be used in an interesting and useful way.

The linear logic approach to planning (see Section 8.2) offers a solution to some of these challenges. However, although parallel composition of actions can be naturally mapped to the commutative $\otimes$ operator of linear logic, sequential composition does not find a natural interpretation. For this reason, for the language I develop in this thesis, I employ systems BV and NEL, respectively, of the calculus of structures which extend multiplicative linear logic and multiplicative exponential linear logic with a non-commutative self-dual logical operator.

1.4. Summary of Results

In this thesis, I address the following interdependent problems, some of which I have already mentioned above:

(*i*) implementations of proof theoretical systems with deep inference,
(*ii*) reducing nondeterminism in proof search with deep inference systems,
(*iii*) design of a common logical language for planning and concurrency.

1.4.1. Implementing Deep Inference. In the calculus of structures, the laws such as associativity and commutativity, which are implicitly imposed on formulae in other formalisms, become explicit by means of equational theories underlying logical systems (see Chapter 2). By establishing a strict correspondence between the systems of the calculus of structures and term rewriting systems modulo equational theories, I implement proof search, in the calculus of structures, for classical logic, linear logic, system BV and system NEL. These implementations are developed in two independent lines: The first one makes use of the simple high level language and the term rewriting features of the Maude system, whereas the second one uses the pattern matching preprocessor TOM, developed in LORIA/INRIA, which is used to integrate term rewriting features into programming languages such as C, Java, and Caml[8].

1.4.2. Reducing Nondeterminism in Proof Search. The deep inference feature of the calculus of structures brings shorter proofs than any other formalism supporting analytical proofs [**Gug04c**]. However, because the inference rules can be applied in many more ways than, for instance, in the sequent calculus, the breadth of the search space in proof search increases (see Chapter 5). In this thesis, I develop a technique which reduces this nondeterminism, and makes these shorter proofs more immediately accessible. I present this technique on system BV. By exploiting the common scheme followed by the systems of the calculus of structures, this technique can be analogously applied to other systems of the calculus of structures. As an evidence to this, I apply this technique to classical logic systems in the calculus of structures in order to obtain equivalent systems where nondeterminism is reduced. Because this technique is strongly related with a cut elimination argument, it remains perfectly clean from a proof theoretical point of view. Furthermore, I use this technique as a combinatoric proof theoretical tool for showing that system BV is NP-complete.

1.4.3. A Common Language for Planning and Concurrency. I present a common logical language for planning and concurrency. Planning is a quite developed field which is very much motivated by logic. However, the research in this field is mainly based on exploiting the expressive power of first order classical logic, in a way motivated by efficiency concerns. In such a perspective, the languages for planning lack the theoretical insight, which would serve to understand the nature of these problems. However concurrency theory, which has similar problems in focus, with a formal perspective, has the potential to provide the theoretical insight for a deeper understanding of the problems being attacked in planning.

In this thesis, I establish the basis of a uniform logical language which aims at bringing planning and concurrency closer. Such a language provides a bridge between these two fields, so that techniques can be carried both ways. For instance, the highly developed formal methods of concurrency theory provide the necessary tools to establish a notion of equivalence of plans. Furthermore, planning techniques can be used for concurrency theoretic queries, for instance, for verification of security protocols.

In the language that I present, I further elaborate on the linear logic approach to planning by resorting to system NEL. System NEL is shown to be Turing-complete.

[8]These implementations are available for download at `http://www.iccl.tu-dresden.de/~ozan/maude_cos.html`

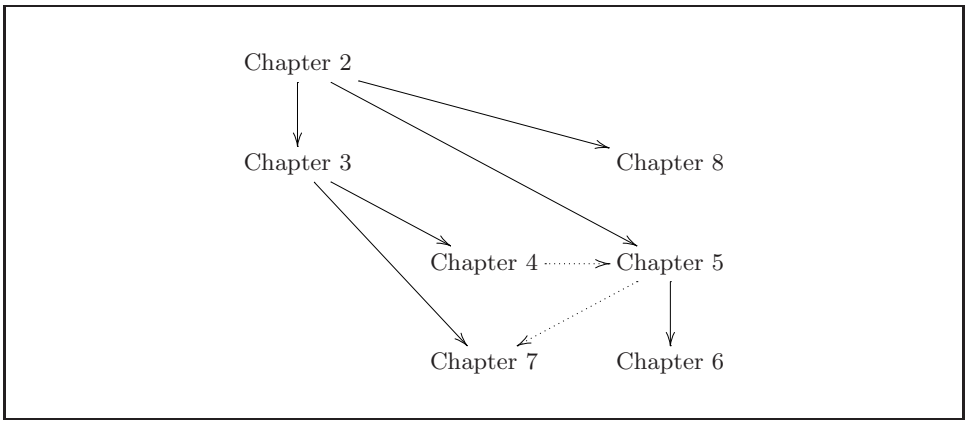


FIGURE 1.2. The dependencies between the chapters

By employing this system, the sequential composition of actions is represented by the self-dual non-commutative logical operator. This way, it becomes possible to express parallel and sequential composition of actions, at the same level, in a purely logical language. The inference rules of system NEL give the operational semantics of this language. In linear logic approach to planning, the plans are extracted from a proof of the specification of the planning problem. In contrast, this language allows to compute a partial order plan as a premise of a derivation, the conclusion of which is the specification of the planning problem (see Chapter 8).

In contrast to plans computed by partial order planners in the literature, due to the explicit treatment of resources, these partial order plans capture the independence and causality between actions. I describe a procedure which delivers a behavioral concurrency semantics of a planning problem specification. This way, I establish a notion of plan equivalence with respect to canonical representation of plans as partial orders.

Some of the results presented in this thesis have already appeared elsewhere [**HK04, Kah04a, Kah04b, Kah05, KMR05a, Kah06b, Kah06a, Kah07**]. This thesis can be read as shown in Figure 1.2. There, the dashed arrows denote the weak dependencies which can be ignored by the reader in a hurry.

CHAPTER 2

Proof Theory with Deep Inference: the Calculus of Structures

In this chapter, I will review the notions and notations of the calculus of structures. For this purpose, I will first introduce system BV, following [**Gug07**], which started the research on the calculus of structures. I will then present a Turing-complete extension of system BV, called system NEL [**GS02**]. These two systems will be of particular importance for the results in Chapter 8 where I introduce a common language for planning and concurrency. Then, I will give a brief introduction to the systems in the calculus of structures for linear logic and classical logic. For an in-depth exposure to the proof theory of these logics, the reader is referred to [**Brü03b**] and [**Str03a**].

The calculus of structures is a proof theoretical formalism which works on *structures*. From a syntactic point of view, structures can be seen as equivalence classes of formulae: The laws such as associativity and commutativity, which are usually implicitly imposed on formulae, become explicit on structures by means of an underlying equational system in a logical system of the calculus of structures. In fact, the intuition behind structures can be observed best in their graphical representation, called *relation webs*, where the mutual logical relations between atoms of a structure are represented by the arcs of a graph. Informally, relation webs are graphs that can be seen as canonical representations of equivalence classes of formulae. Because relation webs are not yet fully developed for other logics than system BV, I will postpone the discussion on the relation webs to Chapter 5, where they will play an important role in proving properties of system BV.

Like formulae, structures are built from atoms. The negation of a structure is denoted by the bar $\bar{\cdot}$. In contrast to common infix notation used for binary connectives, as in formulae, structures are written in *out-fix* notation. For example, talking about classical logic, the structure $[(a, \bar{b}, c), (d, e)]$ corresponds to the formula $(a \wedge \bar{b} \wedge c) \vee (d \wedge e)$. This notation, does not only provide a more algebraic reading of the logical expressions, but also provides a uniform syntax for all the logics, which are studied in the calculus of structures. For instance, when we are working in linear logic, the structure above corresponds to the linear logic formula $(a \otimes \bar{b} \otimes c) \parr (d \otimes e)$. This way, the common behavior in different logics can be better observed, for instance, by means of inference rules which are common to these systems. A typical example is the so called switch rule (see, e.g., Definition 2.17), which is common to all the systems, in the calculus of structures, for all the logics. However, in each system this rule deals with different notions of conjunction and disjunction. The out-fix notation also provides an easier reading of the derivations of the calculus of structures, which are not trees as in most deductive formalisms, but chains of inferences (see, e.g., Subsection 2.1.3).

$$\mathsf{id}\,\frac{}{\vdash A, \bar{A}} \qquad \otimes\,\frac{\vdash A, \Phi \quad \vdash B, \Psi}{\vdash A \otimes B, \Phi, \Psi} \qquad \parr\,\frac{\vdash A, B, \Phi}{\vdash A \parr B, \Phi} \qquad \bot\,\frac{\vdash \Phi}{\vdash \bot, \Phi} \qquad \mathsf{1}\,\frac{}{\vdash \mathsf{1}}$$

FIGURE 2.1. MLL in the sequent calculus

2.1. System BV

System BV [**Gug07**] is an extension of multiplicative linear logic (MLL, see Figure 2.1), with the rules *mix* (mix) and *nullary mix* (mix0, see Figure 2.2), and a self-dual non-commutative logical operator, called *seq*. (For the formal relation between MLL and system BV, see Subsection 2.1.3).

The calculus of structures was originally conceived to introduce system BV in order to capture the sequential composition of process algebras by means of a self-dual, non-commutative logical operator. In fact, Bruscoli showed, in [**Bru02**], that there is a strict correspondence between system BV and a fragment of the process algebra CCS (see, e.g., [**Mil89**]).

2.1.1. Structures. The definition below presents the notion of structure for system BV. The structures for other logics, in the following, are defined similarly, however with slight differences with respect to their different languages.

DEFINITION 2.1. *There are countably many* positive atoms *and countably many* negative atoms. *Atoms are denoted by* $a, b, c, \ldots$ BV structures *are generated by*

$$R ::= \circ \mid a \mid [\,R\,,\,R\,] \mid (\,R\,,\,R\,) \mid \langle\,R\,;\,R\,\rangle \mid \overline{R}$$

where $\circ$*, the* unit*, is not an atom.* $[\,R\,,\,R\,]$ *is called a* par structure*,* $(\,R\,,\,R\,)$ *is called a* copar structure*, and* $\langle\,R\,;\,R\,\rangle$ *is called a* seq structure*.* $\overline{R}$ *is the* negation *of the structure* R*.* Structures *are denoted by* $P, Q, R, S, T, \ldots$*.* BV *structures are considered equivalent modulo the relation* $\approx$*, which is the smallest congruence relation induced by the equational system shown in Figure 2.3.*

REMARK 2.2. *Similar to* BV *structures, structures for other systems are often considered equivalent modulo a relation* $\approx$*, which is the smallest congruence relation induced by a set of equations. Such a smallest congruence relation always exists because the intersection of two congruence relations, induced by the same equational theory, is also a congruence relation. Because the relation* $\approx$ *is a congruence relation the axioms for reflexivity, symmetry, transitivity, and congruence (context closure) are implicitly included in the equational systems underlying systems of the calculus of structures (see Subsection 3.1).*

$$\mathsf{mix}\,\frac{\vdash \Phi \quad \vdash \Psi}{\vdash \Phi, \Psi} \qquad\qquad \mathsf{mix0}\,\frac{}{\vdash}$$

FIGURE 2.2. The rules mix and nullary mix

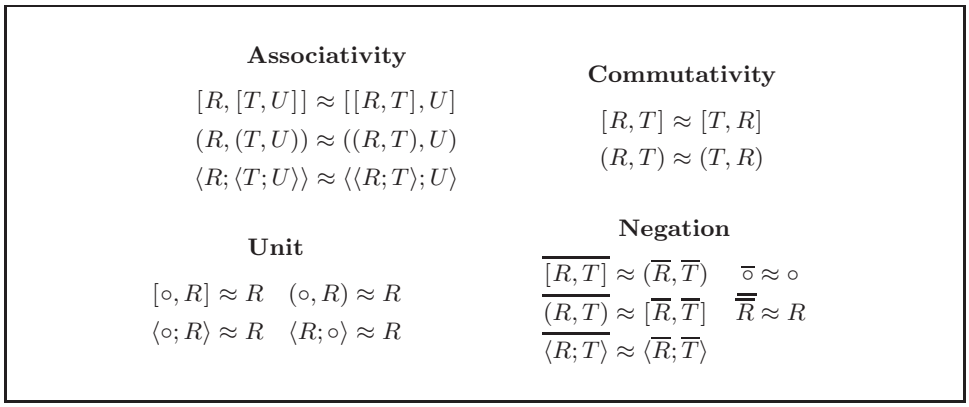


FIGURE 2.3. The equational system underlying BV structures

DEFINITION 2.3. *A* structure context*, denoted by* $S\{\ \}$*, is a structure with one occurence of* $\{\ \}$*, the* empty context *or* hole*. A hole does not appear in the scope of a negation, that is, it does not appear under a negation symbol.* $S\{R\}$ *denotes the structure obtained by filling the hole in* $S\{\ \}$ *with* R*. A structure* R *is a* substructure *of* $S\{R\}$ *and* $S\{\ \}$ *is its* context*.*

NOTATION 2.4. *We drop the context braces if no ambiguity is possible: For instance* $S[R,T]$ *stands for* $S\{[R,T]\}$*.*

DEFINITION 2.5. *A* BV *structure, or a structure context, is* in negation normal form *when the only negated structures appearing in it are atoms; it is* in (unit) normal form *when it is in negation normal form and no unit* $\circ$ *appears in it. If structures* R *and* T *are such that* $R \neq \circ \neq T$*, then the structure* $\langle R;T\rangle$ *is a* proper seq*, the structure* $[R,T]$ *is a* proper par *and the structure* (R,T) *is a* proper copar*. The* BV *structures whose normal forms do not contain seq structures are called* flat*.*

As it can be seen in Figure 2.3, negation obeys the usual De Morgan laws for par and copar, it switches them. The operator seq is self-dual, that is, $\overline{\langle\, S_1 \,;\, S_2 \,\rangle} \approx \langle\, \overline{S_1} \,;\, \overline{S_2} \,\rangle$.

All BV structures can be equivalently considered in normal form, because negation can always be pushed inwards to atoms by using the equations for negation and units can be removed by using the equations for unit. Thus, every BV structure can only be equivalent either to the unit, or to an atom, or, mutually exclusively, to a proper seq, or a proper par, or a proper copar.

NOTATION 2.6. *When denoting structures, I will often use a structure which is in normal form in order to express all the structures that belong to the same equivalence class that is given by the underlying equational system, e.g., for system* BV $\approx$ *in Figure 2.3. Further, when no ambiguity is possible, I will often use n-ary operators as abbreviations for vectors of structures built by binary operators. For instance, the structure* $[[[a,b],c],d]$ *will be denoted by* $[a,b,c,d]$*.*

EXAMPLE 2.7. *Structures* $[a,\circ,b,(\bar{a},\overline{[b,d]})]$ *and* $[a,b,(\bar{a},\bar{b},\bar{d})]$ *are equivalent, and the latter is in normal form. Let* $S\{\ \} = [a,b,(c,\langle\bar{a};\{\ \};\bar{c}\rangle)]$ *and* $R = (b,\bar{d})$*.*

Then $S\{R\} = [a, b, (c, \langle \bar{a}; (b, \bar{d}); \bar{c}\rangle)]$. *The structure* $\langle \circ; [a, b]\rangle$ *is a proper par, since* $[a, b]$ *is a structure in normal form.*

DEFINITION 2.8. *The* size *of a structure or a structure context is the number of atoms appearing in it.*

2.1.2. Rules and Derivations. In this subsection, I will review the notions and notations of the derivations in the calculus of structures and the inference rules of system BV. The definitions of derivations are common to other systems in the calculus of structures that are discussed in this thesis.

DEFINITION 2.9. *In the calculus of structures, an* inference rule *is a scheme of the kind*

$$\rho \frac{T}{R} \quad ,$$

where ρ *is the* name *of the rule,* T *is its* premise *and* U *is its* conclusion. *An inference rule is called an* axiom *if its premise is empty, i.e., the rule is of the shape*

$$\rho \frac{}{R} \quad .$$

REMARK 2.10. *A typical* deep inference rule *has the shape*

$$\rho \frac{S\{T\}}{S\{R\}} \quad .$$

A deep inference rule of this form specifies the logical implication $T \Rightarrow R$ *inside a generic context* $S\{\ \}$*, which is the logical implication being modeled in the system. For system* BV *this is the linear implication* $\multimap$ *(see Subsection 2.1.3).*

DEFINITION 2.11. *An* instance *of an inference rule of the form*

$$\rho \frac{T}{R}$$

is the inference rule ρ *where the structure schemes* R *and* T *are replaced with structures that respect their scheme. When premise and conclusion in an instance of an inference rule are equivalent, that instance is* trivial*, otherwise it is* non-trivial*.*

DEFINITION 2.12. *In an inference rule, following the scheme*

$$\rho \frac{S\{T\}}{S\{R\}} \quad ,$$

the substructure R *is called the* redex *and* T *the* contractum *of the rule's instance.*

EXAMPLE 2.13. *Consider the following trivial instance of the switch rule which is applied inside the structure context* $(\{\ \}, c)$*:*

$$\begin{array}{c} \approx \dfrac{([a, b], c)}{([a, b], \circ, c)} \\ \mathsf{s}\ \dfrac{([[a, \circ), b], c)}{([a, b], c)} \\ \approx \end{array}$$

In the instance of the switch rule above, the structure $[(a, \circ), b]$ *is the redex and the structure* $([a, b], \circ)$*is the contractum.*

DEFINITION 2.14. *A (formal)* system $\mathscr{S}$ *is a set of inference rules.*

DEFINITION 2.15. *A* derivation Δ *in a calculus of structures system is either a structure or a finite chain of instances of inference rules in the system:*

$$\begin{array}{c} \rho\,\dfrac{R}{R'} \\ \rho'\,\dfrac{R'}{\vdots} \\ \rho''\,\dfrac{\vdots}{R''} \end{array}$$

The top-most structure in a derivation, if present, is called the premise *of the derivation, and the bottom-most structure is called its* conclusion*. A derivation* Δ *with premise* T *conclusion* R*, and whose inference rules are in* $\mathscr{S}$ *will be written as*

$$\begin{array}{c} T \\ \Delta \big\| \mathscr{S} \\ R \end{array}.$$

A proof Π *of a structure* R *in the calculus of structures is a derivation whose topmost inference rule is an axiom and the conclusion is the structure* R*. It will be denoted by*

$$\begin{array}{c} \Pi \Big\| \mathscr{S} \\ R \end{array}.$$

The length *of a derivation is the number of instances of inference rules appearing in it.*

DEFINITION 2.16. *A rule* ρ *is* admissible *for a system* $\mathscr{S}$ *if* $\rho \notin \mathscr{S}$ *and for every proof* $\begin{array}{c} \Pi \Big\| \mathscr{S}\cup\{\rho\} \\ R \end{array}$ *there is a proof* $\begin{array}{c} \Pi' \Big\| \mathscr{S} \\ R \end{array}$.

DEFINITION 2.17. *The system* $\{\circ\downarrow, \mathsf{ai}\downarrow, \mathsf{s}, \mathsf{q}\downarrow\}$*, shown in Figure 2.4, is denoted by* BV *and called* basic system V*. The rules of the system are called* unit *(*$\circ\downarrow$*),* atomic interaction *(*$\mathsf{ai}\downarrow$*),* switch *(*s*), and* seq *(*$\mathsf{q}\downarrow$*). The system* $\{\circ\downarrow, \mathsf{ai}\downarrow, \mathsf{s}\}$ *is called* flat system BV*, and denoted by* FBV*.*

DEFINITION 2.18. *The following rule is called* cut *and is denoted by* $\mathsf{ai}\uparrow$*:*

$$\mathsf{ai}\uparrow \frac{S(a, \bar{a})}{S\{\circ\}}$$

THEOREM 2.19. *The cut rule is admissible for system* BV*.*

The proof of the above theorem can be found in [**Gug07**]. However, in Chapter 5, I will prove a similar statement for a system equivalent to system BV.

2.1.3. Relation with the Sequent Calculus. The calculus of structures is a generalization of the sequent calculus. In the calculus of structures, it is possible to design inference rules which correspond to the inference rules of the sequent calculus. However, the deep inference feature of the calculus of structures makes it possible to design deductive systems which are not designable in the sequent

$$\circ\downarrow \frac{}{\circ} \qquad \mathsf{ai}\downarrow \frac{S\{\circ\}}{S[a,\bar{a}]} \qquad \mathsf{s}\,\frac{S([R,U],T)}{S[(R,T),U]} \qquad \mathsf{q}\downarrow \frac{S\langle[R,U];[T,V]\rangle}{S[\langle R;T\rangle,\langle U;V\rangle]}$$

FIGURE 2.4. System BV

calculus. As Tiu showed in [**Tiu01, Tiu06b**], system BV cannot be designed in any standard sequent calculus, because a notion of deep rewriting is necessary in order to get all the provable structures of system BV by means of a deductive system. Below, we will see the relation between system BV and the system MLL of the sequent calculus. The discussions on the relation between the other systems of the calculus of structures, for linear logic and classical logic in the following, and the sequent calculus systems follow the ideas in this subsection.

Inference rules of the sequent calculus can be applied only at the main connective of formulae, i.e., at the connective at the root position when formulae are seen as terms. In contrast, the deep inference feature of the calculus of structures allows the inference rules to access substructures at arbitrary depths inside nested structures. While deep inference plays a crucial role for system BV, it provides systems with a richer proof theory for linear logic and classical logic. In Section 2.3 and Section 2.4, respectively, I will review the calculus of structures presentations of linear logic [**Str03a**] and classical logic [**Brü03b**].

From the point of view of the sequent calculus, structures can be seen as expressions intermediate between formulae and sequents which unify these two entities. In other words, in the calculus of structures, the notions of formula and sequent of the sequent calculus merge into the notion of structure. When we consider BV structures, there is a straight-forward correspondence between flat structures and formulae of multiplicative linear logic (MLL) [**Gir87**].

DEFINITION 2.20. Formulae *are denoted by* $A, B, C, \ldots$ *The* multiplicative linear logic (MLL) formulae *are generated by*

$$A ::= 1 \mid \bot \mid a \mid A \parr A \mid A \otimes A \mid \overline{A}\,.$$

The binary connectives $\parr$ *and* $\otimes$ *are called* par *and* times. $\bar{A}$ *is the* negation *of* A. *Brackets are used to disambiguate expressions when they are necessary. The units* $\bot$ *and* 1, *and the connectives* $\parr$ *and* $\otimes$ *are duals of each other, and they obey the De Morgan laws:*

$$\bar{1} = \bot \qquad \bar{\bot} = 1 \qquad \overline{A \parr B} = \bar{A} \otimes \bar{B} \qquad \overline{A \otimes B} = \bar{A} \parr \bar{B}$$

DEFINITION 2.21. Sequents, *denoted by* Γ, *are expressions of the form*

$$\vdash A_1, \ldots, A_h\,,$$

where $h \geq 0$. *The comma between the formulae* $A_1, \ldots, A_h$ *stands for multiset union. Multisets of formulae are denoted by* Φ *and* Ψ.

DEFINITION 2.22. *The system shown in Figure 2.1 is called* multiplicative linear logic *or system* MLL.

The sequent calculus system for MLL is sometimes extended with the rules mix and mix0 in Figure 2.2 (see, e.g., [**Ret93, FR94, Bel97**]).

DEFINITION 2.23. *The* system MLLx *is the system resulting from extending the system* MLL *in Figure 2.1 with the rules* mix *and* mix0 *in Figure 2.2.*

There is a strict correspondence between system MLLx and system FBV: These two systems prove the syntactic variations of the same logical expressions. However, as we have seen above, the notion of a proof in the sequent calculus is different from the notion of a proof in the calculus of structures. In the calculus of structures, there are only single premise inference rules, thus proofs are chains of instances of inference rules. In contrast, due to multiple premise inference rules of the sequent calculus, which cause branching, the sequent calculus proofs take a tree shape. As we have seen in Chapter 1, because the application of an inference rule is not limited to main connective in the calculus of structures, there are more proofs of a provable logical expression than there is in the sequent calculus.

DEFINITION 2.24. *A* sequent calculus derivation *in a sequent calculus system* $\mathscr{S}$ *is a tree that is represented with*

$$\begin{array}{c} \Gamma_1 \ \cdots \ \Gamma_h \\ \diagdown\!\!\diagup \\ \Gamma \end{array}$$

where $h \geq 0$*, the sequents* $\Gamma_1, \ldots, \Gamma_h$ *are called* premises*,* Γ *is the* conclusion*, and a finite number of instances of the inference rules in system* $\mathscr{S}$ *are applied. A sequent calculus derivation with no premise is a* sequent calculus proof.

To see the correspondence between the flat BV structures and MLL formulae formally, let us have a look at the following definition that I borrow from [**Gug07**].

DEFINITION 2.25. *The function* $\underline{\cdot}_{\mathsf{V}}$ *transforms the formulae of* MLL*, which do not contain the constants* 1 *and* $\bot$ *of linear logic into flat structures according to the following inductive definition:*

$$\underline{a}_{\mathsf{V}} = a \ , \qquad \underline{A \parr B}_{\mathsf{V}} = [\underline{A}_{\mathsf{V}}, \underline{B}_{\mathsf{V}}] \ , \qquad \underline{A \otimes B}_{\mathsf{V}} = (\underline{A}_{\mathsf{V}}, \underline{B}_{\mathsf{V}}) \ .$$

The domain of $\underline{\cdot}_{\mathsf{V}}$ *is extended to sequents as follows:*

$$\underline{\vdash A_1, \ldots, A_h}_{\mathsf{V}} = [\underline{A_1}_{\mathsf{V}}, \ldots, \underline{A_h}_{\mathsf{V}}]$$

EXAMPLE 2.26. *Consider the* MLL *sequent* $\vdash ((a \otimes b) \parr \bar{a}, \bar{b}$ *. By employing the function* $\underline{\cdot}_{\mathsf{V}}$*, we obtain the flat* BV *structure* $[(a, b), \bar{a}, \bar{b}]$*.*

When the deductive system MLL is extended with the rules mix and mix0, some formulae which are not provable in MLL, become provable in MLLx.

EXAMPLE 2.27. *The formula* $a \parr \bar{a} \parr b \parr \bar{b}$ *is not provable in* MLL*, however in* MLLx *it is provable:*

$$\dfrac{\dfrac{\mathsf{Ax}\dfrac{}{\vdash a, \bar{a}}}{\parr\dfrac{}{\vdash a \parr \bar{a}}} \quad \dfrac{\mathsf{Ax}\dfrac{}{\vdash b, \bar{b}}}{\parr\dfrac{}{\vdash b \parr \bar{b}}}}{\mathsf{mix}\dfrac{\vdash a \parr \bar{a}, b \parr \bar{b}}{\parr\dfrac{}{\vdash a \parr \bar{a} \parr b \parr \bar{b}}}}$$

In linear logic, (linear) implication is defined as $A \multimap B = \bar{A} \parr B$. In MLLx it holds that $\mathsf{1} \equiv \bot$, that is, the implications $\bot \multimap \mathsf{1}$ and $\mathsf{1} \multimap \bot$ are provable. Because of this, one can safely map the constants $\mathsf{1}$ and $\bot$ to a single unit, for instance to $\circ$, as in system BV. Theorem below and the Proposition thereafter state that systems FBV and MLLx prove the syntactic variations of the same formulae, and that system BV is a conservative extension of system FBV. The proofs of these results and more detailed discussion of these ideas can be found in [**Gug07**].

THEOREM 2.28. *For any multiplicative linear logic formulae A which does not contain any constants* $\mathsf{1}$ *and* $\bot$*, there is a proof* $\dfrac{}{\vdash A}$ *in* MLLx *if and only if there is a proof* $\dfrac{\Pi \Vdots \mathsf{FBV}}{\vdash A}\,\mathsf{V}$.

DEFINITION 2.29. *A system $\mathscr{S}$ is a* conservative extension *of a system $\mathscr{S}'$, if any provable structure of $\mathscr{S}$, involving symbols of $\mathscr{S}$ only, is provable in $\mathscr{S}'$.*

PROPOSITION 2.30. *System* BV *is a conservative extension of system* FBV*, that is, if a flat structure R is provable in* BV*, then it is also provable in* FBV*.*

Another feature that distinguishes the calculus of structures from the sequent calculus is the presence of an explicit top-down symmetry in the derivations: In the calculus of structures, by flipping a sound derivation upside-down and negating it, one obtains a derivation which is also sound.

This symmetry in the calculus of structures derivations is due to the logical duality between the implications $T \Rightarrow R$ and $\bar{R} \Rightarrow \bar{T}$, which is well known under the name *contrapositive*. For the case of system BV, this implication is the linear implication. Because the inference rules model implications in the logic, in the calculus of structures rules come in pairs of dual rules: A down-version

$$\rho\downarrow \frac{S\{T\}}{S\{R\}}\,,$$

and an up-version which is obtained by expressing the contrapositive of the implication of the down rule as an inference rule, that is,

$$\rho\uparrow \frac{S\{\overline{R}\}}{S\{\overline{T}\}}\,.$$

For instance, the *cut* rule (see Definition 2.18) is the dual of the *atomic interaction* rule (see Definition 2.17).

Because of the syntactic restrictions of the sequent calculus, this kind of duality cannot be observed directly in the sequent calculus, without further proof theoretical analysis of the inference rules. For instance, let us consider the identity rule and the cut rule:

$$\mathsf{id}\ \frac{}{\vdash A, \bar{A}} \qquad\qquad \mathsf{cut}\ \frac{\vdash A, \Phi \quad \vdash \bar{A}, \Psi}{\vdash \Phi, \Psi}$$

It is easy to see that these two rules are not syntactic duals of each other, although their duality can be observed by further proof theoretical analysis, e.g., by means of proof nets [**Gir87**].

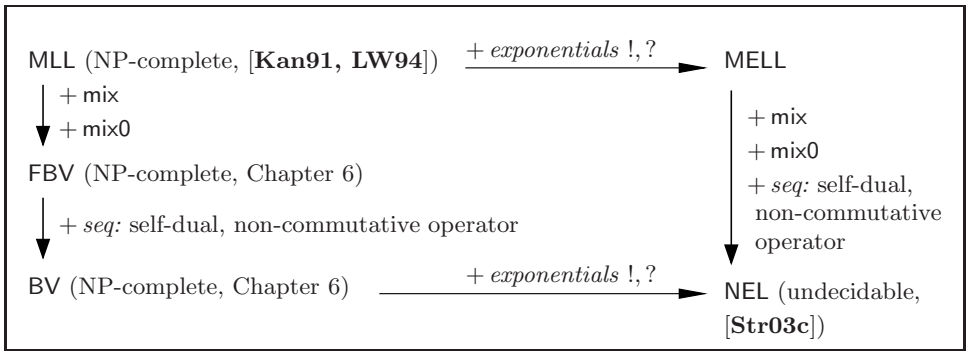


FIGURE 2.5. The relationship between MLL, FBV, BV, MELL and NEL

In a calculus of structures system, all the rules which belong to the up-fragment of the system are admissible: The cut elimination argument modularly generalizes to the whole up-fragment, and this way it becomes possible to eliminate all the up rules (see, e.g., [**Brü03a, Str02**]). In this thesis, I consider in general the down rules, which provide sound and complete systems.

2.2. System NEL

System NEL was introduced by Guglielmi and Straßburger in [**GS02**]. System NEL is a conservative extension of system BV with the exponentials of linear logic. In other words, system NEL is an extension of multiplicative exponential linear logic (see Section 2.3) (MELL) with the rules mix, mix0, and the self-dual non-commutative logical operator seq. The exponentials (! and ?) of linear logic serve to attain controlled contraction and weakening in system NEL. Although it is unknown whether multiplicative exponential linear logic is decidable or not, in [**Str03c**], Straßburger showed that system NEL is undecidable. In Chapter 6, I will show that system BV is NP-complete. Figure 2.5 summarizes the relationship between MLL, FBV, BV, MELL and NEL.

The structures of NEL are generated by a grammar which extends that of BV with the exponentials of linear logic:

DEFINITION 2.31. *Atoms and structures of system* NEL *are denoted as those of system* BV. NEL structures *are generated by*

$$R ::= \circ \mid a \mid [\,R\,,\,R\,] \mid (\,R\,,\,R\,) \mid \langle\,R\,;\,R\,\rangle \mid ?R \mid !R \mid \overline{R}$$

where, in addition to the structures of system BV, $?R$ *is called a* why not structure, *and* $!R$ *is called an* of course structure. NEL *structures are considered equivalent modulo the relation* $\approx$, *which is the smallest congruence relation induced by the equational system shown in Figure 2.6. A* NEL *structure, or a structure context, is* in negation normal form *when the only negated structures appearing in it are atoms; it is* in (unit) normal form *when it is in negation normal form, no unit* $\circ$ *appears in it, and no exponentials* $!, ?$ *can be equivalently eliminated.*

All NEL structures can be equivalently considered in normal form, because negation can always be pushed inwards to atoms by using the equalities for negation,

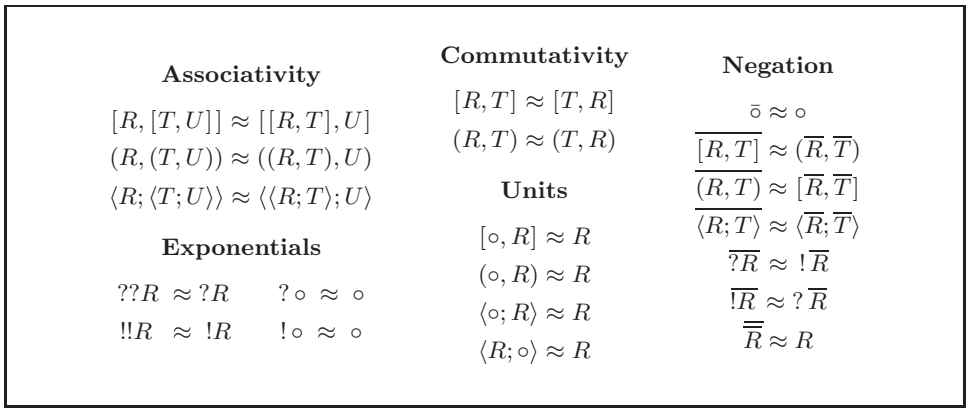


FIGURE 2.6. The equational system underlying NEL structures

and units and redundant exponentials can always be removed by using the equations for units and exponentials.

Similar to the correspondence between BV structures and MLL formulae, there is a straight-forward correspondence between structures not involving seq and MELL formulae. For example $![(?a,b),\bar{a},!\bar{b}]$ corresponds to $!\,((?a\otimes b)\parr\bar{a}\parr!\,\bar{b})$, and vice versa. For a detailed discussion on the proof theory of NEL and the precise relation between NEL and MELL, the reader is referred to [**GS02, Str03a**].

DEFINITION 2.32. *The system in Figure 2.7 is called* non-commutative exponential linear logic*, or system* NEL*. The rules of the system are* unit *(*$\circ\downarrow$*),* atomic interaction *(*$\mathsf{ai}\downarrow$*),* switch *(*s*),* seq *(*$\mathsf{q}\downarrow$*),* promotion *(*$\mathsf{p}\downarrow$*),* weakening *(*$\mathsf{w}\downarrow$*), and* absorption *(*$\mathsf{b}\downarrow$*).*

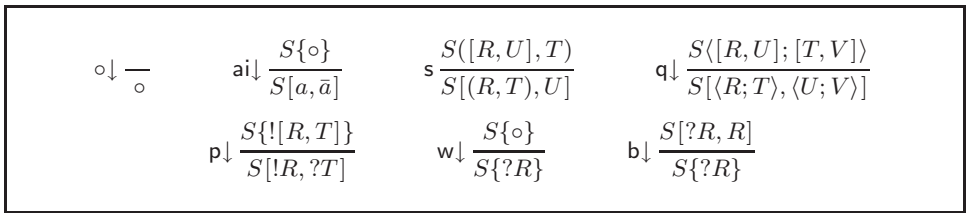


FIGURE 2.7. System NEL

2.3. Linear Logic in the Calculus of Structures

In this section, I will review the calculus of structures presentation of linear logic, i.e., system LS, following [**Str03a**].

DEFINITION 2.33. *In the language of system* LS*, there are countably many* positive atoms *and* negative atoms*, which are denoted by* $a,b,c,\ldots$*, and there are*

four selected atoms $\bot$, $\mathsf{1}$, $\mathsf{0}$, *and* $\top$ *that are called* bottom, one, zero, *and* top, *respectively.* LS structures *are generated by*

$$R ::= \bot \mid \mathsf{1} \mid \mathsf{0} \mid \top \mid a \mid [R, R] \mid (R, R) \mid \{R, R\} \mid \langle R, R \rangle \mid !R \mid ?R \mid \overline{R}$$

where $[R, R]$ *is called a* par structure, (R, R) *is called a* copar (times) structure, $\{R, R\}$ *is called a* plus structure, $\langle R, R \rangle$ *is called a* with structure, $!R$ *is called an* of course structure, *and* $?R$ *is called a* why not structure. $\overline{R}$ *is the* negation *of the structure* R. *The atoms* $\bot$, $\mathsf{1}$, $\mathsf{0}$ *and* $\top$ *are the units for par, times, plus, and with structures, respectively.* LS *structures are considered equivalent modulo the relation* $\approx$, *which is the smallest congruence relation induced by the equational system shown in Figure 2.8. An* LS *structure, or a structure context, is* in negation normal form *when the only negated structures appearing in it are atoms; it is* in (unit) normal form *when it is in negation normal form, and no units and exponentials can be equivalently removed.*

Remark 2.34. *In* [**Str03a**]*, Straßburger defines units as selected atoms. This results in simplifications in the design of the system* LS *and proofs of the results on this system. In this thesis, I use the system* LS *in* [**Str03a**]*, thus I use the convention that units are atoms in system* LS.

All LS structures can be equivalently considered in negation normal form, because negation can always be pushed inwards to atoms by using the equations for negation. Furthermore, by using the equations for unit on a structure in negation normal form, the unit normal formal form of a structure can always be obtained.

Associativity

$$[R,[T,U]] \approx [[R,T],U]$$
$$(R,(T,U)) \approx ((R,T),U)$$
$$\{R,\{T,U\}\} \approx \{\{R,T\},U\}$$
$$\langle R,\langle T,U\rangle\rangle \approx \langle\langle R,T\rangle,U\rangle$$

Units

$$[\bot,R] \approx R \qquad (\mathsf{1},R) \approx R$$
$$\{\mathsf{0},R\} \approx R \qquad \langle\top,R\rangle \approx R$$
$$\{\bot,\bot\} \approx \bot \qquad \langle \mathsf{1},\mathsf{1}\rangle \approx \mathsf{1}$$

Commutativity

$$[R,T] \approx [T,R]$$
$$(R,T) \approx (T,R)$$
$$\{R,T\} \approx (T,R)$$
$$\langle R,T\rangle \approx (T,R)$$

Exponentials

$$??R \approx ?R$$
$$!!R \approx !R$$
$$?\bot \approx \bot$$
$$!\mathsf{1} \approx \mathsf{1}$$

Negation

$$\overline{[R,T]} \approx (\bar{R},\bar{T})$$
$$\overline{(R,T)} \approx [\bar{R},\bar{T}]$$
$$\overline{\{R,T\}} \approx \langle\bar{R},\bar{T}\rangle$$
$$\overline{\langle R,T\rangle} \approx \{\bar{R},\bar{T}\}$$
$$\overline{?R} \approx !\bar{R}$$
$$\overline{!R} \approx ?\bar{R}$$
$$\bar{\bar{R}} \approx R$$
$$\bar{\mathsf{1}} \approx \bot \qquad \bar{\bot} \approx \mathsf{1}$$
$$\bar{\top} \approx \mathsf{0} \qquad \bar{\mathsf{0}} \approx \top$$

Figure 2.8. The equational system underlying LS structures

We are now ready to give the calculus of structures presentation of linear logic:

Definition 2.35. *The system* $\{\mathsf{1}\downarrow, \mathsf{ai}\downarrow, \mathsf{s}, \mathsf{t}\downarrow, \mathsf{c}\downarrow, \mathsf{d}\downarrow, \mathsf{w}\downarrow, \mathsf{b}\downarrow, \mathsf{p}\downarrow\}$*, shown in Figure 2.9, is called* linear logic *in the calculus of structures, or system* LS. *The rules of the system are called* one *(*$\mathsf{1}\downarrow$*),* atomic interaction *(*$\mathsf{ai}\downarrow$*),* switch *(*s*),* promotion *(*$\mathsf{p}\downarrow$*),* weakening *(*$\mathsf{w}\downarrow$*),* absorption *(*$\mathsf{b}\downarrow$*),* thinning *(*$\mathsf{t}\downarrow$*),* contraction *(*$\mathsf{c}\downarrow$*), and* additive *(*$\mathsf{d}\downarrow$*).*

DEFINITION 2.36. *The fragment of system* LS *which consists of the rules* $\mathsf{1}\downarrow, \mathsf{ai}\downarrow,$ *and* s *is called* multiplicative linear logic in the calculus of structures*, or* system S*.*

DEFINITION 2.37. *The rules* $\mathsf{p}\downarrow, \mathsf{w}\downarrow,$ *and* $\mathsf{b}\downarrow$ *are called* the exponential rules*. The fragment of system* LS *which consists of system* S *together with the exponential rules is called* multiplicative exponential linear logic in the calculus of structures*, or* system ELS*.*

DEFINITION 2.38. *The rules* $\mathsf{t}\downarrow, \mathsf{w}\downarrow,$ *and* $\mathsf{d}\downarrow$ *are called* the additive rules*. The fragment of system* LS *which consists of system* S *together with the additive rules is called* multiplicative additive linear logic in the calculus of structures*, or* system ALS*.*

$$\mathsf{1}\downarrow \frac{}{1} \qquad \mathsf{ai}\downarrow \frac{S\{1\}}{S[a,\bar{a}]} \qquad \mathsf{s}\frac{S([R,U],T)}{S[(R,T),U]} \quad \Big\} \text{ System } \mathsf{S}$$

$$\mathsf{p}\downarrow \frac{S\{![R,T]\}}{S[!R,?T]} \qquad \mathsf{w}\downarrow \frac{S\{\bot\}}{S\{?R\}} \qquad \mathsf{b}\downarrow \frac{S[?R,R]}{S\{?R\}} \quad \Big\} \text{ Exponential rules}$$

$$\mathsf{t}\downarrow \frac{S\{0\}}{S\{R\}} \qquad \mathsf{c}\downarrow \frac{S\{R,R\}}{S\{R\}} \qquad \mathsf{d}\downarrow \frac{S(\langle R,T\rangle,\langle U,V\rangle)}{S\langle(R,T),[U,V]\rangle} \quad \Big\} \text{ Additive rules}$$

FIGURE 2.9. System LS

Let me now review the language of LL following [**Gir87**]. The language of LL extends that of MLL:

DEFINITION 2.39. *The* linear logic (LL) formulae *are generated by*

$$A ::= 1 \mid \bot \mid 0 \mid \top \mid a \mid A \parr A \mid A \otimes A \mid A \oplus A \mid A \,\&\, A \mid ?A \mid !A \mid \overline{A}\ .$$

The binary connectives $\parr$*,* $\otimes$*,* $\oplus$*, and* $\&$ *are called* par*,* times*,* plus*, and* with*, respectively.* $\bar{A}$ *is the* negation *of* A*.* $?$ *and* $!$ *are modalities, and they are called* of-course *and* why-not*, respectively. Brackets are used to disambiguate expressions when they are necessary. The units* $\bot$ *and* 1*,* 0 *and* $\top$*, the connectives* $\parr$ *and* $\otimes$*,* $\oplus$ *and* $\&$*, and modalities* $?$ *and* $!$ *are duals of each other, and they obey the De Morgan laws:*

$$\bar{1} = \bot \qquad \bar{\bot} = 1 \qquad \overline{A \parr B} = \bar{A} \otimes \bar{B} \qquad \overline{A \otimes B} = \bar{A} \parr \bar{B}$$
$$\bar{0} = \top \qquad \bar{\top} = 0 \qquad \overline{A \oplus B} = \bar{A} \,\&\, \bar{B} \qquad \overline{A \,\&\, B} = \bar{A} \oplus \bar{B}$$
$$\overline{!A} = ?\bar{A} \qquad \overline{?A} = !\bar{A}$$

In order to see that system LS is complete for linear logic, let us have a look at the following definition that I borrow from [**Str03a**].

DEFINITION 2.40. *The function* $\underline{\cdot}_{\mathsf{S}}$ *transforms the linear logic formulae into* LS *structures according to the following inductive definition:*

$$\begin{array}{rclcrclcrcl}
\underline{\bot}_{\mathsf{S}} & = & \bot & , & \underline{A \parr B}_{\mathsf{S}} & = & [\underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}] & , & \underline{a}_{\mathsf{S}} & = & a \quad , \\
\underline{1}_{\mathsf{S}} & = & 1 & , & \underline{A \otimes B}_{\mathsf{S}} & = & (\underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}) & , & \underline{?\,A}_{\mathsf{S}} & = & ?\,\underline{A}_{\mathsf{S}} \quad , \\
\underline{0}_{\mathsf{S}} & = & 0 & , & \underline{A \oplus B}_{\mathsf{S}} & = & \{\underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}\} & , & \underline{!\,A}_{\mathsf{S}} & = & !\,\underline{A}_{\mathsf{S}} \quad , \\
\underline{\top}_{\mathsf{S}} & = & \top & , & \underline{A \,\&\, B}_{\mathsf{S}} & = & \langle \underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}} \rangle & , & \underline{\bar{A}}_{\mathsf{S}} & = & \overline{\underline{A}_{\mathsf{S}}} \quad .
\end{array}$$

The domain of $\underline{\cdot}_{\mathsf{S}}$ *is extended to sequents as follows:*

$$\begin{array}{rcl}
\underline{\vdash}_{\mathsf{S}} & = & \bot \quad \textit{and} \\
\underline{\vdash A_1, \ldots, A_h}_{\mathsf{S}} & = & [\underline{A_1}_{\mathsf{S}}, \ldots, \underline{A_h}_{\mathsf{S}}] \quad , \quad \textit{for } h \geq 0 \, .
\end{array}$$

DEFINITION 2.41. *The* system LL*, i.e.,* linear logic in the one-sided sequent calculus *is shown in Figure 2.10.*

THEOREM 2.42. [**Gir87**] (Cut Elimination) *Every proof* Π *of a sequent* $\vdash \Phi$ *in system* LL *can be transformed into a cut-free proof* Π'*, i.e., a proof in system* LL *that does not contain an instance of the rule* cut*.*

$$\mathsf{id}\,\frac{}{\vdash A, \bar{A}} \qquad \mathsf{cut}\,\frac{\vdash A, \Phi \quad \vdash \bar{A}, \Psi}{\vdash \Phi, \Psi}$$

$$\otimes\,\frac{\vdash A, \Phi \quad \vdash B, \Psi}{\vdash A \otimes B, \Phi, \Psi} \qquad \parr\,\frac{\vdash A, B, \Phi}{\vdash A \parr B, \Phi} \qquad \bot\,\frac{\vdash \Phi}{\vdash \bot, \Phi} \qquad 1\,\frac{}{\vdash 1}$$

$$\&\,\frac{\vdash A, \Phi \quad \vdash B, \Phi}{\vdash A \,\&\, B, \Phi} \qquad \oplus_1\,\frac{\vdash A, \Phi}{\vdash A \oplus B, \Phi} \qquad \oplus_2\,\frac{\vdash B, \Phi}{\vdash A \oplus B, \Phi} \qquad \top\,\frac{}{\vdash \top, \Phi}$$

$$?\mathsf{d}\,\frac{\vdash A, \Phi}{\vdash ?A, \Phi} \qquad ?\mathsf{c}\,\frac{\vdash ?A, ?A, \Phi}{\vdash ?A, \Phi} \qquad ?\mathsf{w}\,\frac{\vdash \Phi}{\vdash ?A, \Phi} \qquad !\,\frac{\vdash\ A, ?B_1, \ldots, ?B_n}{\vdash !A, ?B_1, \ldots, ?B_n}$$

FIGURE 2.10. System LL in the sequent calculus

THEOREM 2.43. *For a linear logic formulae* A*, there is a proof* $\begin{array}{c} \bigtriangledown \\ \vdash A \end{array}$ *in* LL *if and only if there is a proof* $\begin{array}{c} \Pi \\ \underline{\vdash A}_{\mathsf{S}} \end{array}$ *in* LS *.*

The proof of this theorem and more detailed discussion of the proof theory of system LS can be found in [**Str03a**]. However, in Subsection 4.3.3, I will prove a similar statement for a system which is obtained from system LS by removing the equations for unit from the equational system underlying system LS.

Associativity

$$[R,[T,U]] \approx [[R,T],U]$$
$$(R,(T,U)) \approx ((R,T),U)$$

Units

$$(\mathsf{ff},\mathsf{ff}) \approx \mathsf{ff} \qquad [\mathsf{tt},\mathsf{tt}] \approx \mathsf{tt}$$
$$[\mathsf{ff},R] \approx \mathsf{ff} \qquad (\mathsf{tt},R) \approx \mathsf{tt}$$

Commutativity

$$[R,T] \approx [T,R]$$
$$(R,T) \approx (T,R)$$

Negation

$$\overline{[R,T]} \approx (\overline{R},\overline{T}) \qquad \overline{\mathsf{tt}} \approx \mathsf{ff}$$
$$\overline{(R,T)} \approx [\overline{R},\overline{T}] \qquad \overline{\mathsf{ff}} \approx \mathsf{tt}$$
$$\overline{\overline{R}} \approx R$$

FIGURE 2.11. The equational system underlying KSg structures

2.4. Classical Logic in the Calculus of Structures

In this section, I will given an overview of a calculus of structures presentation of classical logic, i.e., system KSg, following [**Brü03b**].

DEFINITION 2.44. *In the language of system* KSg*, there are countably many* positive atoms *and* negative atoms *which are denoted by a, b, c,...* KSg structures *are generated by*

$$R ::= \mathsf{ff} \mid \mathsf{tt} \mid a \mid [\,R\,,\,R\,] \mid (\,R\,,\,R\,) \mid \overline{R}$$

where ff *and* tt *are the units false and true, respectively.* $[R,R]$ *is a* disjunction *and* (R,R) *is a* conjunction. $\overline{R}$ *is the* negation *of the structure* R. KSg *structures are considered equivalent modulo the relation* $\approx$ *which is the smallest congruence relation induced by the equational system shown in Figure 2.11. A* KSg *structure, or a structure context, is* in negation normal form *when the only negated structures appearing in it are atoms; it is* in (unit) normal form *when it is in negation normal form, and no units can be equivalently removed.*

All KSg structures can be equivalently considered in negation normal form, because negation can always be pushed inwards to atoms by using the equations for negation. Furthermore, by using the equations for unit on a structure in negation normal form, the unit normal formal form of a structure can always be obtained.

DEFINITION 2.45. *The system shown in Figure 2.12, is called classical logic in the calculus of structures, or system* KSg. *The rules of the system are called* axiom *(*$\mathsf{tt}{\downarrow}$*),* atomic interaction *(*$\mathsf{ai}{\downarrow}$*),* switch *(*s*),* contraction *(*$\mathsf{c}{\downarrow}$*), and* weakening *(*$\mathsf{w}{\downarrow}$*).*

DEFINITION 2.46. *The* classical logic formulae *are generated by*

$$A ::= \mathsf{tt} \mid \mathsf{ff} \mid a \mid A \wedge A \mid A \vee A \mid \overline{A}\ .$$

The binary connectives $\wedge$ *and* $\vee$ *are called* conjunction *and* disjunction*, respectively.* $\bar{A}$ *is the* negation *of* A*. Brackets are used to disambiguate expressions when they are necessary. The units* tt *and* ff*, and the connectives* $\wedge$ *and* $\vee$ *are duals of each other, and they obey the De Morgan laws:*

$$\bar{\mathsf{tt}} = \mathsf{ff} \qquad \bar{\mathsf{ff}} = \mathsf{tt} \qquad \overline{A \wedge B} = \bar{A} \vee \bar{B} \qquad \overline{A \vee B} = \bar{A} \wedge \bar{B}$$

$$\mathsf{tt}\downarrow\frac{}{\mathsf{tt}} \qquad \mathsf{ai}\downarrow\frac{S\{\mathsf{tt}\}}{S[a,\bar{a}]} \qquad \mathsf{s}\frac{S([R,U],T)}{S[(R,T),U]} \qquad \mathsf{w}\downarrow\frac{S\{\mathsf{ff}\}}{S\{R\}} \qquad \mathsf{c}\downarrow\frac{S[R,R]}{S\{R\}}$$

FIGURE 2.12. System KSg

The following definition, which I borrow from [**Brü03b**], demonstrates the relationship between classical logic formulae and KSg structures:

DEFINITION 2.47. *The function* $\underline{\cdot}_{\mathsf{C}}$ *transforms the classical logic formulae into* KSg *structures according to the following inductive definition:*

$$\underline{\top}_{\mathsf{C}} = \mathsf{tt} \quad , \quad \underline{A\wedge B}_{\mathsf{C}} = [\underline{A}_{\mathsf{C}},\underline{B}_{\mathsf{C}}] \quad , \quad \underline{a}_{\mathsf{C}} = a \quad ,$$
$$\underline{\bot}_{\mathsf{C}} = \mathsf{ff} \quad , \quad \underline{A\vee B}_{\mathsf{C}} = (\underline{A}_{\mathsf{C}},\underline{B}_{\mathsf{C}}) \quad , \quad \underline{\bar{A}}_{\mathsf{C}} = \overline{\underline{A}_{\mathsf{C}}} \quad .$$

The domain of $\underline{\cdot}_{\mathsf{C}}$ *is extended to sequents as follows:*

$$\underline{\vdash}_{\mathsf{C}} = \mathsf{ff} \quad \textit{and}$$
$$\underline{\vdash A_1,\ldots,A_h}_{\mathsf{C}} = [\underline{A_1}_{\mathsf{C}},\ldots,\underline{A_h}_{\mathsf{C}}] \quad , \quad \textit{for } h\geq 0 \; .$$

$$\top\frac{}{\vdash\top} \qquad \mathsf{R}\wedge\frac{\vdash A,\Phi \quad \vdash B,\Psi}{\vdash\Phi,\Psi,A\wedge B} \qquad \mathsf{RC}\frac{\vdash\Phi,A,A}{\vdash\Phi,A}$$

$$\mathsf{Ax}\frac{}{\vdash A,\overline{A}} \qquad \mathsf{R}\vee\frac{\vdash\Phi,A,B}{\vdash\Phi,A\vee B} \qquad \mathsf{RW}\frac{\vdash\Phi}{\vdash\Phi,A}$$

FIGURE 2.13. $\mathsf{GS1p}$: Sequent calculus system for classical logic

Now let us see a one-sided sequent calculus system for classical logic which is very similar to system KSg. In [**Brü03b**], Brünler shows that proofs in this system, which is also known as *Gentzen-Schütte* system [**TS96**], can be translated into proofs in system KSg, and vice versa. Thus, system KSg is sound and complete for classical propositional logic.

DEFINITION 2.48. *The sequent calculus system shown in Figure 2.13 is called* system $\mathsf{GS1p}$.

THEOREM 2.49. *For a classical logic formulae* A*, there is a proof* $\begin{array}{c}\vdots\\ \vdash A\end{array}$ *in system* $\mathsf{GS1p}$ *if and only if there is a proof* $\begin{array}{c}\Pi\\ \vdash \underline{A}_{\mathsf{C}}\end{array}$ KSg .

The proof of this theorem and more detailed discussion of the proof theory of system KSg can be found in [**Brü03b**].

2.5. Other Systems in the Calculus of Structures

In this chapter, we have seen a brief overview of some systems of the calculus of structures, namely the systems BV, NEL, LS and KSg. An important observation on these systems is that all these systems follow a scheme in which two of the three rules of system BV, namely atomic interaction ($\mathsf{ai}{\downarrow}$) and switch (s), are common to all the systems. For instance, these two rules give the multiplicative fragment of linear logic, whereas system KSg is obtained by adding the contraction and weakening rules to these two rules. Furthermore, the third rule in system BV, which is responsible for the non-commutative context management, is also common to system NEL.

Apart from the systems that I discussed in this chapter, there are other systems in the calculus of structures, which address different computational and proof theoretical properties, also for other logics. In [**BT01**], Brünnler and Tiu introduce a local system for classical logic. Brünnler gives an atomic cut-elimination proof for this system in [**Brü03a**]. In [**Tiu06a**], Tiu presents a local system in the calculus of structures for intuitionistic logic. Straßburger presents a system, in [**Str02**], for linear logic where all the rules are local. Hein and Stewart [**HS05**], and Stewart and Stouppa [**SS05**] present systems for a class of modal logics. All these systems follow a common scheme described in [**Gug02**]. A number of publications, on these logics, and others, also on topics related to deep inference, are available at the calculus of structures web-site[1].

[1] `http://alessio.guglielmi.name/res/cos/`

CHAPTER 3

Deep Inference as Term Rewriting

In the sequent calculus, because of the two-premise inference rules, the derivations are tree-shaped. For instance, let us consider the following sequent calculus rules, which are responsible for context management in the one-sided sequent calculus systems for classical logic and linear logic, respectively:

$$\wedge \frac{\vdash A, \Phi \quad \vdash B, \Psi}{\vdash A \wedge B, \Phi, \Psi} \qquad \otimes \frac{\vdash A, \Phi \quad \vdash B, \Psi}{\vdash A \otimes B, \Phi, \Psi}$$

During the application of such rules in a bottom-up proof construction episode the derivations branch and, this way, take a tree shape. In contrast to *object level*, which is given by the logical connectives, the empty space between the two branches (and the commas in the sequents) in such derivations belong to the so called *meta level* of the proof theoretical system. For some logics, there is a strict correspondence between the meta level branching and the object level of the deductive system. For instance, in the systems for classical logic in the sequent calculus, the branching corresponds to conjunction. On the other hand for some other logics, such as linear logic, there is a mismatch between the meta level and the object level of the deductive system in the sequent calculus. This mismatch is due to the fact that the meta level branching in the proof theoretical system cannot be mapped to a unique logical operator of these logics. For instance, in linear logic for some rules the branching corresponds to the multiplicative conjunction, and in some others it corresponds to the additive conjunction. For more information on this mismatch see [**Gug03**].

The branching in the sequent calculus derivations plays a crucial role in proof construction: While going up in the derivation, the branching allows to partition the formula being proved into smaller formulae, and this way allows to access the subformulae for further applications of the inference rules. Because these inference rules can be applied only at the main connective, this partitioning is crucial for reaching the subformulae while constructing the proofs.

Often such inference rules are implemented in Prolog as bottom-up proof search instructions. For instance, the rule $\wedge$ above can be implemented in Prolog as follows:

```
prove(F):-
       match(F,[A /\ B, M, N]),
       prove([A,M]),
       prove([B,N]).
```

In the sequent calculus, the laws such as associativity and commutativity are implicitly imposed on the sequents and formulae. Thus, in an implementation of a

sequent calculus system these laws should be expressed explicitly. In an implementation following the scheme above, the predicate `match` would then implement the matching of the sequents and formulae under associativity and commutativity.

Because of the branching in the derivations and the applicability of the inference rules only at the main connective, it is not possible to give an immediate computational interpretation of the sequent calculus systems as term rewriting systems. In several approaches that consider sequent calculus from a computation as proof search point of view, the meta-level, which causes branching is expressed by introducing new operators: In [**Dep00, Dep02**], Deplagne introduces the associative commutative operator $\bullet$, which turns the double-premise inference rules into single premise inference rules. For instance, above $\wedge$ rule takes the shape

$$\wedge \frac{\vdash A, \Phi \ \bullet \vdash B, \Psi}{\vdash A \wedge B, \Phi, \Psi} \quad .$$

This way, Deplagne views the sequent calculus systems as term rewriting systems, which can be implemented in the language ELAN [**BKK$^+$98, KK04**] by resorting to associative commutative term rewriting features of this language. However, in this approach the application of the term rewriting rules, which correspond to the inference rules, is restricted: Because the sequent calculus inference rules can be applied only at the main connective, these rewriting rules can be applied only at the root position of the term-tree that represents the formula being proved.

A similar approach was considered, in [**MOM96**], by resorting to rewriting logic [**Mes92**] and its implementation language Maude [**CDE$^+$02**]: There, Martí-Olliet and Meseguer give an encoding of the sequent calculus presentation of linear logic in rewriting logic. For this purpose, they introduce an operator called *configuration*: This operator maps the meta level branching in the sequent calculus proofs to a syntactic expression in rewriting logic [**Mes92**]. By employing this operator, they encode the inference rules of the sequent calculus as top-down rewrite rules in rewriting logic. For instance, the rule $\otimes$ above becomes the following rewriting rule:

```
rl    (|- A,M)  (|- B,N)
   => --- -------------- Tensor
        |- A * B, M, N
```

The authors provide a correctness proof of their encoding, however their approach was considered at a higher level, but not as an implementation of an executable tool.[1]

One of the reasons for this is their choice of top-down view of the inference rules, rather than bottom-up view, which corresponds to computation as proof search view of the inference rules. However, this is not a restriction because these rewriting rules can be similarly represented as bottom-up proof search rules. In [**VMO03**], Verdejo and Martí-Oliet argue along these lines for obtaining executable versions of the ideas presented in [**MOM96**]. Although in some cases it is quite easy, in others this is not the case.

One of the difficulties of implementing the inference rules of the sequent calculus presentation of linear logic as term rewriting rules is due to the following *promotion*

[1]Maude was not available as a running system when [**MOM96**] was published.

rule:

$$\frac{\vdash A, ?B_1, \ldots, ?B_n}{\vdash\, !A, ?B_1, \ldots, ?B_n}\,!$$

Expressing this rule as a straight-forward term rewriting rule is not possible because this rule requires global knowledge of the context of $!A$, that is, the application of this rule requires each formula in the context of $!A$ to be checked to have the form $?B$ for all the $?B_1, \ldots, ?B_n$. However, there is no bound on n in this rule.

In order to get over this problem, in [**MOM96**], Martí-Oliet and Meseguer express this rule as the so called *storage* rule:

```
rl
    |- ?M, A
 => --- ------
    |- ?M, !A
```

In this rule, `M` is a multiset of formulae and `A` is a formula. The operator `?` is extended to multisets by means of axioms

```
? null  =  null
? (M,N) = (? M, ? N)
```

and the rules are applied modulo these axioms among others.

In the calculus of structures, because what is meta level in the sequent calculus is represented at the object level of the deductive systems, proofs are chains of inferences rather than trees. For instance, the role played by the above $\wedge$ and $\otimes$ rules is captured by two consequent applications of the switch rule:

$$\mathsf{s}\frac{([A,M],[B,N],[R,T])}{([[(A,M],B),N],[R,T])}$$
$$\mathsf{s}\frac{}{([[(A,B),M,N],[R,T])}$$

However, in contrast to the inference rules of the sequent calculus, which can be applied only at the main connective, the inference rules of the calculus of structures can be applied at any depth inside a structure as in term rewriting: For instance, in the above derivation, the inference rules are applied inside the context

$$(\,\{\quad\},[R,T])\,.$$

Furthermore, in the calculus of structures, the promotion rule is replaced with the following rule which does not require a global view of the structures:

$$\mathsf{p}\!\downarrow\frac{S\{![R,T]\}}{S[!R,?T]}$$

These observations suggests a correspondence between the term rewriting systems and the deductive systems of the calculus of structures. In the following, exploiting this observation, I will present a procedure that turns derivations in the calculus of structures into rewritings in a term rewriting system. Because the structures of a deductive system of the calculus of structures are considered equivalent modulo an equational theory, the term rewriting relation that I employ is modulo equational theories. I will present this procedure on system BV, and then generalize it to other systems of the calculus of structures.

In this thesis, I will consider the systems from a bottom-up (analytical) point of view, such that the conclusion is the starting point of a derivation and inference

rules are used to reach the desired premises. However, the top-down point of view of the derivations can be analogously considered.

3.1. Term Rewriting: Basic Definitions

In this section, I collect basic definitions for terms, positions, replacements, substitutions, equations and rewrite rules as can be found in, e.g., [**BN98**] or [**Pla93**]. The reader familiar with these notions may skip this section.

DEFINITION 3.1. *A* signature Σ *is a set of* function symbols*, where each* $f \in \Sigma$ *is associated with a non-negative integer* n*, the arity of* f*. For* $n \geq 0$*, we denote the set of all* n*-ary elements of* Σ *by* $\Sigma^{(n)}$*. The elements of* $\Sigma^{(0)}$ *are called* constant symbols*.*

EXAMPLE 3.2. *Consider the signature which I will use to denote addition on non-negative integers:* $\Sigma = \{e, i, f\}$*, where* e *has arity* 0*,* i *is unary, and* f *is binary. Talking about the set of non-negative integers,* e *denotes the smallest non-negative integer, and* i *denotes the successor function. The function symbol* f *denotes addition on this set.*

DEFINITION 3.3. *Given a signature* Σ *and a set* $\mathcal{V}$ *of variables with* $\Sigma \cap \mathcal{V} = \emptyset$*, the set* $T(\Sigma, \mathcal{V})$ *of all* Σ*-terms over* $\mathcal{V}$ *is inductively defined as*

- $\mathcal{V} \subseteq T(\Sigma, \mathcal{V})$ *(i.e., every variable is a term),*
- *for all* $n \geq 0$*, all* $f \in \Sigma^{(n)}$*, and all* $t_1, \ldots, t_n \in T(\Sigma, \mathcal{V})$*, we have* $f(t_1, \ldots, t_n) \in T(\Sigma, \mathcal{V})$ *(i.e., application of function symbols to terms yields terms).*

EXAMPLE 3.4. *For the signature* $\Sigma = \{e, i, f\}$ *above,* $f(e, f(x, i(x)))$ *is a* Σ*-term that contains the variable* x*, whereas* $f(e)$ *is not a* Σ*-term because* f *is binary function symbol.*

DEFINITION 3.5. *Let* Σ *be a signature,* $\mathcal{V}$ *be a set of variables, and* $s \in T(\Sigma, \mathcal{V})$*.*

(1) *The set of* positions $\mathsf{pos}(s)$ *of a term* s *is inductively defined as follows:*
   - *If* $s = X \in \mathcal{V}$*, then* $\mathsf{pos}(s) = \{\Lambda\}$*.*
   - *If* $s = f(s_1, \ldots, s_n)$*, then* $\mathsf{pos}(s) = \{\Lambda\} \cup \bigcup_{i=1}^{n} \{\, i\gamma \mid \gamma \in \mathsf{pos}(s_i) \,\}$*.*

   The position Λ *is called the* root position *of the term* s*, and the function or variable symbol at this position is called the* root symbol*.*

(2) *For* $\gamma \in \mathsf{pos}(s)$*, the* sub-term of s at position γ*, denoted by* $s|_\gamma$*, is inductively defined as follows:*
   - $s|_\Lambda = s$*.*
   - $f(s_1, \ldots, s_n)|_{i\gamma} = s_i|_\gamma$*.*

(3) *For* $\gamma \in \mathsf{pos}(s)$*, the term obtained from* s *by* replacing the sub-term at position γ by t*, denoted by* $s|t|_\gamma$*, is inductively defined as follows:*
   - $s|t|_\Lambda = t$*.*
   - $f(s_1, \ldots, s_n)|t|_{i\gamma} = f(s_1, \ldots, s_i|t|_\gamma, \ldots, s_n)$*.*

EXAMPLE 3.6. *For the term* $s = f(e, f(x, i(x)))$*,* $\mathsf{pos}(s) = \{\Lambda, 1, 2, 21, 22, 221\}$*,* $s|_{22} = i(x)$*,* $s|e|_2 = f(e, e)$*.*

DEFINITION 3.7. *Let* Σ *be a signature, and* $\mathcal{V}$ *be a countably infinite set of variables. A* substitution σ *is a mapping from the set* $\mathcal{V}$ *of variables to the set* $T(\Sigma, \mathcal{V})$ *of* Σ*-terms, which is equal to the identity except for finitely many variables. Thus,* σ *can be represented by* $\{X \mapsto \sigma(X) \mid \sigma(X) \neq X\}$*.* ε *denotes the* empty

substitution. *The set of variables that σ does not map to themselves is called the* domain of σ*: $Dom(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. The* range *of σ is $Ran(\sigma) = \{\sigma(x) \mid x \in Dom(\sigma)\}$. The* instance *of a term s with respect to σ, denoted by $\sigma(s)$ or $s\sigma$, is the term obtained by simultaneously replacing each occurrence of variables from $Dom(\sigma)$ in s by the corresponding term in $Ran(\sigma)$.*

EXAMPLE 3.8. *Let $s = f(e, x)$ and $t = f(y, f(x, y))$, and let $\sigma = \{x \mapsto i(y), y \mapsto e\}$. Then $\sigma(s) = f(e, i(y))$ and $\sigma(t) = f(e, f(i(y), e))$.*

DEFINITION 3.9. *Let Σ be a signature, and $\mathcal{V}$ be a countably infinite set of variables. An* equation *is an expression of the form $s \approx t$, where s and t are Σ-terms. An* equational system *is a set of equations. We implicitly assume that the equational axioms, i.e., the axioms for reflexivity, symmetry, transitivity, and substitutivity are added to each equational system. Let $\approx_{\mathsf{E}}$ be the smallest congruence relation induced by an equational system E.*

As mentioned in Chapter 2, a smallest congruence relation induced by an equational system always exists because the intersection of two congruence relations, induced by the same equational system, is a congruence relation.

EXAMPLE 3.10. *Consider the equational system*

$$\mathsf{E} = \{\, f(x, y) \approx f(y, x),\ f(x, f(y, z)) \approx f(f(x, y), z)\} \,,$$

which denotes the commutativity and associativity of f.

DEFINITION 3.11. *A* rewrite rule *is an expression of the form $l \to r$, where l is a non-variable term and r is a term. A* term rewriting system *is a set of rewrite rules. A* redex *is an instance of a left-hand side of a rewrite rule. Given terms s, t and a term rewriting system R, s* rewrites to *t* with respect to *R, denoted by $s \to_{\mathsf{R}(\rho,\gamma,\sigma)} t$ if there is a position $\gamma \in \mathsf{pos}(s)$ and a substitution σ such that $s|_\gamma = \sigma(l)$ and $t = s|\sigma(r)|_\gamma$, where ρ is the rewrite rule being applied. In this case, we say $s|_\gamma$* matches *l. We drop the subscript (ρ, γ, σ) when no ambiguity is possible.* Contracting *a redex means replacing it by the corresponding instance of the right-hand side of the rule. $\overset{n}{\to}$ denotes the n-fold composition of $\to$: $\overset{0}{\to}$ is the identity relation. Where $n \geq 0$, $s \overset{n+1}{\to} t$ is defined as, for some t', $s \overset{n}{\to} t' \to t$. $\overset{*}{\to}$ denotes the reflexive transitive closure of $\to$. For a term s and term rewriting system R, s is* in normal form with respect to *R, if there is no term t such that $s \to_{\mathsf{R}} t$. Two terms s and t are* joinable *if there is a term u such that $s \overset{*}{\to} u \overset{*}{\leftarrow} t$.*

EXAMPLE 3.12. *Let R be the term rewriting system with the rules*

$$f(x, e) \to x \ \text{ and } f(x, i(y)) \to i(f(x, y)) \,.$$

The term $s = f(z, i(e))$ rewrites to $i(f(z, e))$ with respect to R because, for $\sigma = \{x \mapsto z, y \mapsto e\}$, we have $s|\,\sigma(\,i(f(x, y))\,)\,|_\Lambda = i(f(z, e))$.

DEFINITION 3.13. *Given terms s, t, a term rewriting system R and an equational system E, s* rewrites to *t* with respect to *R* and *E, denoted by $s \to_{\mathsf{R}/\mathsf{E}(\rho,\gamma,\sigma)} t$ if there are terms s', t', a rewrite rule $\rho = l \to r$, a position $\gamma \in \mathsf{pos}(s')$ and a substitution σ such that $s \approx_{\mathsf{E}} s'$, $s'|_\gamma = \sigma(l)$, $t' = s'|\sigma(r)|_\gamma$ and $t' \approx_{\mathsf{E}} t$. For an equational system E, the term rewriting system R* modulo *E will indicate the term rewriting system R such that the rules of R are applied with respect to rewrite relation R/E.*

We can rephrase the definition of the rewrite relation R/E above in other words as follows:

$$s \to_{\mathsf{R}/\mathsf{E}(\rho,\gamma,\sigma)} t \quad \text{if and only if} \quad (\exists s', t')\, s \approx_{\mathsf{E}} s' \ \wedge\ s' \to_{\mathsf{R}(\rho,\gamma,\sigma)} t' \ \wedge\ t' \approx_{\mathsf{E}} t\,.$$

Example 3.14. *Let* R *be the term rewriting system in Example 3.12 and* E *be the equational system in Example 3.10. Then we have* $f(i(e), i(i(e))) \to_{\mathsf{R}/\mathsf{E}} i(f(i(i(e)), e))$ *because*

$$f(i(e), i(i(e))) \approx_{\mathsf{E}} f(i(i(e)), i(e)) \to_{\mathsf{R}} i(f(i(i(e)), e)) \approx_{\mathsf{E}} i(f(i(i(e)), e))\,.$$

The rewrite relation R, E, introduced in [**PS81**], is another rewriting relation for rewriting modulo equality. However, this relation is weaker than the relation R/E, and it is not feasible for the purpose of this chapter (see Section 3.7).

Definition 3.15. *A term rewriting system* R *is* terminating *if for any term* t_0 *there is no infinite descending chain* $t_0 \to_{\mathsf{R}} t_1 \to_{\mathsf{R}} \cdots$. *It is* confluent *if, for any term* t, $t_2 \overset{*}{\leftarrow} t \overset{*}{\rightarrow} t_1$ *implies that there is a term* t_3 *such that* $t_2 \overset{*}{\rightarrow} t_3 \overset{*}{\leftarrow} t_1$.

3.2. Replacing Equivalence Classes with Equality Steps

In the calculus of structures, the structures are considered equivalent modulo an equational system. Because of this, the inference rules are generally considered to be applied to equivalence classes of structures. However, such a point of view of the structures does not provide a specification of an explicit operational definition of the application of the inference rules. In this section, in a first step for an explicit operational definition of a derivation and, in particular, for an operational definition of the application of an inference rule, I will make the role played by the syntactic equations in a derivation explicit. For this purpose, I will separate the notion of a structure from the equivalence class defined by the syntactic equations of the structures: Each derivation step between two equivalence classes will be split into three steps: An equality step leading to a new element of the first equivalence class, then an application of an inference rule to this element, and then another equality step leading to an element of the second equivalence class. Hence, in this chapter, from this point on, a structure as a syntactic object will denote an element of an equivalence class of structures, but not the equivalence class itself.

Definition 3.16. *For system* $\mathscr{S} \in \{\mathsf{BV}, \mathsf{NEL}, \mathsf{LS}, \mathsf{KSg}\}$, *let* $\approx$ *be the smallest congruence relation on* $\mathscr{S}$ *structures induced by the equations underlying system* $\mathscr{S}$. *A structure* R *is a* derivation from R to R *in system* $\mathscr{S}$. *If* Δ *is a derivation from structure* R *to structure* T, $T \approx T'$, *there is an instance of an inference rule* ρ *with conclusion* T' *and premise* Q', *and* $Q' \approx Q$ *then*

$$\begin{array}{c} \approx \dfrac{Q}{Q'} \\ \rho\,\dfrac{T'}{T} \\ \approx \dfrac{T}{\Delta\,\Big\|\ R} \end{array}$$

is a derivation from R to Q *in system* $\mathscr{S}$*. The notion of a proof is analogously redefined: If* Δ *is a derivation from* R *to* T *and* $T \approx \circ$*, then*

$$\begin{array}{c} \circ\!\downarrow \dfrac{}{\circ} \\ \approx \dfrac{}{T} \\ \Delta \Big\| \\ R \end{array}$$

is a proof *of* R*.*

Example 3.17. *Consider the proof in* BV *on the left-hand side below, which is replaced with the proof on the right-hand side, where the structure expressions do not represent equivalence classes of structures, but members of the equivalence class:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\circ}\,\circ\!\downarrow}{[c,\bar{c}]}\,\mathsf{ai}\!\downarrow}{[(c,[\bar{b},b]),\bar{c}]}\,\mathsf{ai}\!\downarrow}{[[b,(c,\bar{b})],\bar{c}]}\,\mathsf{s}}{[\langle[\bar{a},a];[b,(\bar{b},c)]\rangle,\bar{c}]}\,\mathsf{ai}\!\downarrow}{[\langle\bar{a};b\rangle,\langle a;(\bar{b},c)\rangle,\bar{c}]}\,\mathsf{q}\!\downarrow \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\circ}\,\circ\!\downarrow}{\circ}\approx}{[c,\bar{c}]}\,\mathsf{ai}\!\downarrow}{[(\circ,c),\bar{c}]}\approx}{[([b,\bar{b}],c),\bar{c}]}\,\mathsf{ai}\!\downarrow}{[([\bar{b},b],c),\bar{c}]}\approx}{[[(\bar{b},c),b],\bar{c}]}\,\mathsf{s}}{[\langle\circ;[b,(\bar{b},c)]\rangle,\bar{c}]}\approx}{[\langle[a,\bar{a}];[b,(\bar{b},c)]\rangle,\bar{c}]}\,\mathsf{ai}\!\downarrow}{[\langle[\bar{a},a];[b,(\bar{b},c)]\rangle,\bar{c}]}\approx}{[[\langle\bar{a};b\rangle,\langle a;(\bar{b},c)\rangle],\bar{c}]}\,\mathsf{q}\!\downarrow}{[\langle\bar{a};b\rangle,[\langle a;(\bar{b},c)\rangle,\bar{c}]]}\approx \quad .$$

In the above proofs, not all the structures are in normal form. As before, at an application of an inference rules, the holes at which inference rules are applied are not under the scope of negation.

Because $\approx$ is the smallest congruence relation induced by the equational system shown in Figure 2.3, each derivation and each proof as defined in Chapter 2 can be transformed into a derivation and a proof as defined in this section, respectively. Thus the role of the equational theory underlying derivations is clarified from the point of view of an operational definition of the inferences. Similar ideas have been considered also in [**Brü03b**].

3.3. Replacing Structures with Terms

In this section, I will replace the notion of a structure with the notion of a term. This way, I will consider variables over terms, thus formalizing the concept of structures with variable occurrences.

Definition 3.18. *Let* Σ_{BV} *be the signature given by*

$$\{\,\circ,\ \bar{\_},\ [\_,\_],\ (\_,\_),\ \langle\_;\_\rangle\,\} \cup \{a \mid a \text{ is an atom}\}\,.$$

Then, BV *structures as defined in Section 2.1 are* Σ_{BV}*-terms over the empty set of variables, i.e., ground* Σ_{BV}*-terms. On the other hand, by considering a non-empty set* $\mathcal{V}$ *of variables, we obtain* Σ_{BV}*-terms over* $\mathcal{V}$*. The notions of* Σ_{NEL}*-terms,* Σ_{LS}*-terms, and* Σ_{KSg}*-terms are defined analogously with respect to following signatures:*

$$\Sigma_{\mathsf{NEL}} = \{\circ,\ \overline{\_},\ [\_,\_],\ (\_,\_),\ \langle \_;\_\rangle,\ ?\_,\ !\_\} \cup \{a \mid a \text{ is an atom}\};$$

$$\Sigma_{\mathsf{LS}} = \{\bot, \mathsf{1}, \mathsf{0}, \top, \overline{\_},\ [\_,\_],\ (\_,\_),\ \{\_,\_\},\ \langle\_,\_\rangle,\ ?\_,\ !\_\} \cup \{a \mid a \text{ is an atom}\};$$

$$\Sigma_{\mathsf{KSg}} = \{\mathsf{tt}, \mathsf{ff}, \overline{\_},\ [\_,\_],\ (\_,\_)\} \cup \{a \mid a \text{ is an atom}\}.$$

From now on, for a system $\mathscr{S}$ in the calculus of structures, I will use the notions $\mathscr{S}$ structure and $\Sigma_{\mathscr{S}}$-term synonymously.

3.4. Replacing Contexts with Positions

In this section, I will replace the notion of a structure context with the notion of a position. This will provide a precise operational specification of which substructure or sub-term is being replaced in a derivation step.

As structures are terms the notions introduced in Section 3.1 can be applied.

EXAMPLE 3.19. *Let* $s = [[(\bar{b}, c), b], \bar{c}]$ *and* $t = ([\bar{b}, b], c)$ *then*

$$\mathsf{pos}(s) = \{\Lambda,\ 1,\ 11,\ 111,\ 1111,\ 112,\ 12,\ 2,\ 21\}$$

and

$$s|t|_1 = [([\bar{b}, b], c), \bar{c}].$$

Thus, the notion of positions, sub-terms and the replacement of a sub-term by another one at a particular position take over the role of a structure context.

3.5. Orienting the Equations for Negation

The definition of negation normal form of structures corresponds to a standard definition on formulae in the literature. From an operational point of view, considering the negation normal form of a formula is advantageous: Given that each application of an inference rule yields again a formula in negation normal form, the syntactic equivalences concerning negation can be removed from the underlying theory.

An inspection of the systems BV, NEL, LS, and KSg shows that in a bottom-up application of an inference rule no new negation signs are introduced: The only inference rule which involves negation in these systems is the atomic interaction rule. In a bottom-up application of this rule the negated atoms get annihilated.[2] Furthermore, holes in a structure do not appear in the scope of a negation sign. Thus, the property that negated structures appear only in atoms is preserved by the application of the inference rules.

This observation points out the possibility of orienting the equations for negation as rewrite rules from left to right in order to get the negation normal form at the very beginning of a bottom-up construction of a derivation. Because these rules

[2]In this thesis, I consider the inference rules from a bottom-up point of view. However, if the atomic interaction rule is considered from a top-down point of view, because the negation is on atoms, but not on generic structures, restricting ourselves to structures in negation normal form would not cause any problems.

do not introduce any new negation signs, neither when they are applied bottom-up nor top-down, the negation normal form of the structures is preserved by the application of the inference rules through out the construction of the derivation.

DEFINITION 3.20. *The term rewriting system* $\mathsf{R}^{\mathsf{BV}}_{Neg}$, *obtained by orienting the equations for negation in Figure 2.3 from left to right, is defined as follows:*

$$\mathsf{R}^{\mathsf{BV}}_{Neg} = \left\{ \begin{array}{lcl@{\qquad}lcl} \overline{[R,T]} & \to & (\bar{R},\bar{T}) & \overline{(R,T)} & \to & [\bar{R},\bar{T}] \\ \overline{\langle R;T\rangle} & \to & \langle \bar{R};\bar{T}\rangle & \bar{\bar{R}} & \to & R \\ \bar{\circ} & \to & \circ & & & \end{array} \right.$$

DEFINITION 3.21. *The term rewriting system* $\mathsf{R}^{\mathsf{NEL}}_{Neg}$, *obtained by orienting the equations for negation in Figure 2.6 from left to right, is defined as follows:*

$$\mathsf{R}^{\mathsf{NEL}}_{Neg} = \mathsf{R}^{\mathsf{BV}}_{Neg} \cup \left\{ \begin{array}{lcl} \overline{?R} & \to & !\bar{R} \\ \overline{!R} & \to & ?\bar{R} \end{array} \right.$$

DEFINITION 3.22. *The term rewriting system* $\mathsf{R}^{\mathsf{LS}}_{Neg}$, *obtained by orienting the equations for negation in Figure 2.8 from left to right, is defined as follows:*

$$\mathsf{R}^{\mathsf{LS}}_{Neg} = \left\{ \begin{array}{lcl@{\qquad}lcl} \overline{[R,T]} & \to & (\bar{R},\bar{T}) & \overline{\{R,T\}} & \to & \langle \bar{R},\bar{T}\rangle \\ \overline{(R,T)} & \to & [\bar{R},\bar{T}] & \overline{\langle R,T\rangle} & \to & \{\bar{R},\bar{T}\} \\ \overline{?R} & \to & !\bar{R} & \overline{!R} & \to & ?\bar{R} \\ \bar{1} & \to & \bot & \bar{\top} & \to & 0 \\ \bar{\bot} & \to & 1 & \bar{0} & \to & \top \\ \bar{\bar{R}} & \to & R & & & \end{array} \right.$$

DEFINITION 3.23. *The term rewriting system* $\mathsf{R}^{\mathsf{KSg}}_{Neg}$, *obtained by orienting the equations for negation in Figure 2.11 from left to right, is defined as follows:*

$$\mathsf{R}^{\mathsf{KSg}}_{Neg} = \left\{ \begin{array}{lcl@{\qquad}lcl} \overline{[R,T]} & \to & (\bar{R},\bar{T}) & \bar{\bar{R}} & \to & R \\ \overline{(R,T)} & \to & [\bar{R},\bar{T}] & \bar{\mathtt{tt}} & \to & \mathtt{ff} \\ & & & \bar{\mathtt{ff}} & \to & \mathtt{tt} \end{array} \right.$$

PROPOSITION 3.24. *Term rewriting systems* $\mathsf{R}^{\mathsf{BV}}_{Neg}$, $\mathsf{R}^{\mathsf{NEL}}_{Neg}$, $\mathsf{R}^{\mathsf{LS}}_{Neg}$, *and* $\mathsf{R}^{\mathsf{KSg}}_{Neg}$ *are*

(*i*) *terminating;*
(*ii*) *confluent.*
(*iii*) *Let* s *be a* Σ_{BV}*-term (*Σ_{NEL}*-term,* Σ_{LS}*-term,* Σ_{KSg}*-term, respectively). The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{BV}}_{Neg}$ *(*$\mathsf{R}^{\mathsf{NEL}}_{Neg}$, $\mathsf{R}^{\mathsf{LS}}_{Neg}$, $\mathsf{R}^{\mathsf{KSg}}_{Neg}$, *respectively,) is in negation normal form.*

PROOF. (*i*) It suffices to take a lexicographic path order as stated in [**BN98**]:

- for $\mathsf{R}^{\mathsf{BV}}_{Neg}$, take
$$\bar{\_} >_{lpo} [\_,\_] >_{lpo} (\_,\_) >_{lpo} \langle \_;\_\rangle >_{lpo} \circ\,;$$
- for $\mathsf{R}^{\mathsf{NEL}}_{Neg}$, take
$$\bar{\_} >_{lpo} ?\_ >_{lpo} !\_ >_{lpo} [\_,\_] >_{lpo} (\_,\_) >_{lpo} \langle \_;\_\rangle >_{lpo} \circ\,;$$

- for $\mathsf{R}^{\mathsf{LS}}_{Neg}$, take

$$\bar{\_} >_{lpo} ?\_ >_{lpo} !\_ >_{lpo} [\_,\_] >_{lpo} (\_,\_) >_{lpo} \{\_,\_\} >_{lpo} \langle\_,\_\rangle >_{lpo} \bot >_{lpo} \mathsf{1} >_{lpo} \top >_{lpo} \mathsf{0}\,;$$

- for $\mathsf{R}^{\mathsf{KSg}}_{Neg}$, take $\bar{\_} >_{lpo} [\_,\_] >_{lpo} (\_,\_) >_{lpo} \mathsf{tt} >_{lpo} \mathsf{ff}$.

(*ii*) For $\mathscr{S} \in \{\mathsf{BV}, \mathsf{NEL}, \mathsf{LS}, \mathsf{KSg}\}$ because $\mathsf{R}^{\mathscr{S}}_{Neg}$ is terminating, the result follows from the analysis of the critical pairs: The proof for other systems being similar, let us see the case for $\mathscr{S} = \mathsf{NEL}$: For two rewriting rules $l_1 \to r_1$ and $l_2 \to r_2$ which have mutually distinct variables, let $\gamma \in \mathsf{pos}(l_1)$ such that $l_1|_\gamma$ is not a variable. If $l_1|_\gamma$ and l_2 are unifiable with a most general unifier σ, then the pair $(\sigma r_1\,,\, \sigma l_1|\sigma r_2|_\gamma)$ determines a critical pair. The only rule that has a non-variable sub-term of the left-hand-side which is unifiable with the left-hand-side of another rule is the rule $\bar{\bar{R}} \to R$. We have the critical pairs $([R,T], \overline{(\bar{R},\bar{T})})$, $((R,T), \overline{[\bar{R},\bar{T}]})$, $(\langle R;T\rangle, \overline{\langle \bar{R};\bar{T}\rangle})$, $(?\,R, \overline{!\,\bar{R}})$, $(!\,R, \overline{?\,\bar{R}})$, and $(\circ, \bar{\circ})$ which are joinable.

(*iii*) For $\mathscr{S} \in \{\mathsf{BV}, \mathsf{NEL}, \mathsf{LS}, \mathsf{KSg}\}$, s being in negation normal form and applicability of a rewrite rule of $\mathsf{R}^{\mathscr{S}}_{Neg}$ are contradictory. □

Remark 3.25. *In the systems of the calculus of structures, which are discussed in this thesis (and also in others), the inference rules do not introduce any new negation symbols on generic structures. With the above proposition, it is possible to disregard the equations for negation in the systems of the calculus of structures by considering the structures that are in negation normal form. In the rest of the thesis, by assuming that the negation normal form of the structures are obtained by employing the above term rewriting systems, I will disregard the equations for negation. However, I will often generalize the notion of negation normal form to other normal forms by extending the above term rewriting systems by other rewrite rules.*

3.6. Replacing Inference Rules with Rewrite Rules

In this section, I will define term rewriting systems that correspond to the bottom-up view of the inference rules of the systems of the calculus of structures. For this purpose, I will first define the term rewriting system RBV which corresponds to system BV, and then analogously extend this definition to other systems.

Definition 3.26. *Each inference rule occurring in* BV *as shown in Figure 2.4 except* $\circ\downarrow$ *is turned into a rewrite rule as shown in Figure 3.1 by dropping the context* S*. We consider* $\mathsf{ai}\downarrow$ *to be a schema for all atoms* a*.*

Definition 3.27. *Let* EBV *be the equational system obtained by removing the equations for negation from the equations in Figure 2.3.*

By employing the rewrite relation R/E of Definition 3.13, we can now compute rewrite sequences as follows:

EXAMPLE 3.28. *The following rewrite sequence corresponds precisely to the proof given in Example 3.17.*

$$
\begin{array}{ll}
[\langle \bar{a}; b\rangle, [\langle a; (\bar{b}, c)\rangle, \bar{c}]] & \\
\approx_{\mathsf{EBV}} & [[\langle \bar{a}; b\rangle, \langle a; (\bar{b}, c)\rangle], \bar{c}] \\
\rightarrow_{\mathsf{RBV}(\mathsf{q}\downarrow, 1, \{R \mapsto \bar{a},\, R' \mapsto b,\, T \mapsto a,\, T' \mapsto (\bar{b}, c)\})} & [\langle [\bar{a}, a]; [b, (\bar{b}, c)]\rangle, \bar{c}] \\
\approx_{\mathsf{EBV}} & [\langle [a, \bar{a}]; [b, (\bar{b}, c)]\rangle, \bar{c}] \\
\rightarrow_{\mathsf{RBV}(\mathsf{ai}\downarrow, 11, \varepsilon)} & [\langle \circ; [b, (\bar{b}, c)]\rangle, \bar{c}] \\
\approx_{\mathsf{EBV}} & [[(\bar{b}, c), b], \bar{c}] \\
\rightarrow_{\mathsf{RBV}(\mathsf{s}, 1, \{R \mapsto \bar{b},\, T \mapsto c,\, U \mapsto b\})} & [([\bar{b}, b], c), \bar{c}] \\
\approx_{\mathsf{EBV}} & [([b, \bar{b}], c), \bar{c}] \\
\rightarrow_{\mathsf{RBV}(\mathsf{ai}\downarrow, 11, \varepsilon)} & [(\circ, c), \bar{c}] \\
\approx_{\mathsf{EBV}} & [c, \bar{c}] \\
\rightarrow_{\mathsf{RBV}(\mathsf{ai}\downarrow, \Lambda, \varepsilon)} & \circ
\end{array}
$$

$[a, \bar{a}]$	$\rightarrow$	$\circ$	$\mathsf{ai}\downarrow$
$[(R, T), U]$	$\rightarrow$	$([R, U], T)$	s
$[\langle R; T\rangle, \langle U; V\rangle]$	$\rightarrow$	$\langle [R, U]; [T, V]\rangle$	$\mathsf{q}\downarrow$

FIGURE 3.1. The rewrite system RBV corresponding to BV

Because the systems BV, NEL, LS, and KSg share a common scheme with respect to the ideas above, we can apply the above ideas to these other systems:

DEFINITION 3.29. *Each inference rule occurring in* NEL *as shown in Figure 2.7 except* $\circ\downarrow$ *is turned into a rewrite rule as shown in Figure 3.2 by dropping the context* S*. We consider* $\mathsf{ai}\downarrow$ *to be a schema for all atoms* a*.*

DEFINITION 3.30. *Let* ENEL *be the equational system obtained by removing the equations for negation from the equations in Figure 2.6.*

$[a, \bar{a}]$	$\rightarrow$	$\circ$	$\mathsf{ai}\downarrow$
$[(R, T), U]$	$\rightarrow$	$([R, U], T)$	s
$[\langle R; T\rangle, \langle U; V\rangle]$	$\rightarrow$	$\langle [R, U]; [T, V]\rangle$	$\mathsf{q}\downarrow$
$[?R, !T]$	$\rightarrow$	$![R, T]$	$\mathsf{p}\downarrow$
$?R$	$\rightarrow$	$\circ$	$\mathsf{w}\downarrow$
$?R$	$\rightarrow$	$[?R, R]$	$\mathsf{b}\downarrow$

FIGURE 3.2. The rewrite system RNEL corresponding to NEL

DEFINITION 3.31. *Each inference rule occurring in* LS *as shown in Figure 2.9 except* $1\downarrow$ *is turned into a rewrite rule as shown in Figure 3.3 by dropping the context* S*. We consider* $\mathsf{ai}\downarrow$ *to be a schema for all atoms* a*.*

DEFINITION 3.32. *Let* ELS *be the equational system obtained by removing the equations for negation from the equations in Figure 2.8.*

$[a, \bar{a}]$	$\to$	1	$\mathsf{ai}{\downarrow}$
$[(R,T),U]$	$\to$	$([R,U],T)$	s
$[?R, !T]$	$\to$	$![R,T]$	$\mathsf{p}{\downarrow}$
$?R$	$\to$	$\bot$	$\mathsf{w}{\downarrow}$
$?R$	$\to$	$[?R,R]$	$\mathsf{b}{\downarrow}$
R	$\to$	0	$\mathsf{t}{\downarrow}$
R	$\to$	$\{R,R\}$	$\mathsf{c}{\downarrow}$
$[\langle R,T\rangle, \{U,V\}]$	$\to$	$\langle [R,U],[T,V]\rangle$	$\mathsf{d}{\downarrow}$

FIGURE 3.3. The rewrite system RLS corresponding to LS

DEFINITION 3.33. *Each inference rule occurring in* KSg *as shown in Figure 2.12 except* $\mathsf{tt}{\downarrow}$ *is turned into a rewrite rule as shown in Figure 3.4 by dropping the context* S*. We consider the rule* $\mathsf{ai}{\downarrow}$ *to be a schema for all atoms* a*.*

DEFINITION 3.34. *Let* EKSg *be the equational system obtained by removing the equations for negation from the equations in Figure 2.11.*

$[a, \bar{a}]$	$\to$	tt	$\mathsf{ai}{\downarrow}$
$[(R,T),U]$	$\to$	$([R,U],T)$	s
R	$\to$	ff	$\mathsf{w}{\downarrow}$
R	$\to$	$[R,R]$	$\mathsf{c}{\downarrow}$

FIGURE 3.4. The rewrite system RKSg corresponding to KSg

PROPOSITION 3.35. *For* $\mathscr{S} \in \{\mathsf{BV}, \mathsf{NEL}, \mathsf{LS}, \mathsf{KSg}\}$*, let* s *and* t *be* $\Sigma_{\mathscr{S}}$*-terms or structures which are in negation normal form.*

1. *There is a derivation in* $\mathscr{S}$ *from* s *to* t *having length* n *if and only if there exists a rewriting* $s \overset{n}{\to}_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t$*.*
2. *There is a proof of* s *in* $\mathscr{S}$ *having length* n *if and only if there exists a rewriting*
   - $s \overset{n-1}{\to}_{\mathsf{RBV/EBV}} \circ$ *if* $\mathscr{S} = \mathsf{BV}$*.*
   - $s \overset{n-1}{\to}_{\mathsf{RNEL/ENEL}} \circ$ *if* $\mathscr{S} = \mathsf{NEL}$*.*
   - $s \overset{n-1}{\to}_{\mathsf{RLS/ELS}} 1$ *if* $\mathscr{S} = \mathsf{LS}$*.*
   - $s \overset{n-1}{\to}_{\mathsf{RKSg/EKSg}} \mathsf{tt}$ *if* $\mathscr{S} = \mathsf{KSg}$*.*

PROOF. The proof of (1) is by induction on the length of the derivation in $\mathscr{S}$ and the number of rewrite steps in $\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}$, respectively, for the if part and the only if part, respectively: For the base case, observe that there is a derivation in $\mathscr{S}$ with

length 0, that is there is a $\Sigma_{\mathscr{S}}$-term or a $\mathscr{S}$ structure s if and only if $s \xrightarrow{0}_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} s$. For the inductive case, single out the top most rule instance ρ, with the premise t and conclusion t', in the derivation from s to t. For the derivation from s to t' with length n, by the induction hypothesis, there is a rewriting $s \xrightarrow{n}_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t'$, construct a rewriting $s \xrightarrow{n}_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t' \rightarrow_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t$ with length $n+1$. For the only if direction, single out the last rewiriting $t' \rightarrow_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t$ in $s \xrightarrow{n+1}_{\mathsf{R}\mathscr{S}/\mathsf{E}\mathscr{S}} t'$, and construct a derivation from s to t with length $n+1$ analogously. (2) follows immediately from (1) with the following observation: The top-most rule instance in proofs in system BV (NEL, LS, KSg, respectively) is an axiom, thus from a proof with length n we can obtain a derivation with length $n-1$ with the premise $\circ$ ($\circ$, $\mathsf{1}$, $\mathtt{tt}$,respectively). □

It is important to observe that in the rewritings generated by this proposition correspond one-to-one to the inference steps in derivations of the corresponding systems.

3.7. Discussion

The main purpose of this chapter was to bring the (rather obvious) relationship between systems of the calculus of structures and term rewriting systems to formal grounds. Establishing this connection does not only allow to observe the inference rules operationally, but also prepares the ground for applications where proof theoretical techniques of the calculus of structures and term rewriting techniques can be applied harmoniously. This result also shows that the techniques of term rewriting can assist the proof theoretical developments on deep inference.

To summarize, in this chapter, we have seen that the structures of the calculus of structures can be expressed as terms over a signature of function symbols denoting logical connectives and atoms, and the derivations and proofs in the proof theoretical systems of the calculus of structures can be expressed as rewritings of term rewriting systems modulo equality. These results can be analogously generalized to other systems of the calculus of structures, since all these systems follow a common scheme which is exploited in this chapter.

Besides the deep inference rules which find a natural interpretation as term rewriting rules, in the calculus of structures it is also possible to design deductive systems with inference rules that resemble the rules of the sequent calculus. Let us call such inference rules, that can be applied only at the root position, i.e., position Λ, shallow (non-deep) rules. Such shallow rules can be expressed as term rewriting rules by introducing a new function symbol that plays the role of turnstile of the sequents. This is similar to the approaches for expressing sequent calculus rules as term rewriting rules, e.g., in [**MOM96, Dep00**]. As an example, consider the shallow version of the s rule:

$$\mathsf{s}' \frac{([R,T],U)}{[(R,U),T]}$$

This rule is a shallow rule because there is no context given in this rules. Such a rule corresponds to a rewrite rule which can only be applied at the root position. We can impose this restriction by introducing a unary function symbol, e.g., "$\vdash \_$", which will be the outer most function symbol of a structure. Then the above rule can be put as the following rewrite rule:

$$\vdash [(R,T),U] \quad \rightarrow \quad \vdash ([R,U],T) \qquad\qquad \mathsf{s}'$$

This way, the inference rules of the calculus of structures can be implemented as non-deep (shallow) inference rules which resemble the sequent calculus inference rules.

For rewriting modulo equational theories, an alternative to rewrite relation R/E is the rewrite relation R,E which was introduced in [**PS81**]. Following [**BN98**], this relation is defined as follows: $s \to_{\mathsf{R},\mathsf{E}\,(\rho,\gamma,\sigma)} t$ if there is a rewrite rule $\rho = l \to r$, a position $\gamma \in \mathsf{pos}(s')$ and a substitution σ such that $s|_\gamma \approx_\mathsf{E} \sigma(l)$ and $t = s|\sigma(r)|_\gamma$.

In rewriting with respect to rewrite relation R,E, each rewriting step involves matching modulo $\approx_\mathsf{E}$, which is weaker than $\to_{\mathsf{R}/\mathsf{E}}$. From the point of view of the calculus of structures systems, this rewrite relation is not feasible. For example, consider the following rewriting step with respect to rewrite relation R/E where $\mathsf{R} = \{\, [(R,T),U] \to ([R,U],T) \,\}$ and $\mathsf{E} = \mathsf{EBV}$:

$$[[(a,b),c],d] \;\approx_\mathsf{E}\; [[(a,b),d],c] \;\to_\mathsf{R}\; [([a,d],b),c] \;\approx_\mathsf{E}\; [([a,d],b),c]$$

This rewriting step corresponds to a bottom-up application of the switch rule in system BV, however such a rewriting is not possible with respect to rewrite relation R,E because there is no position γ and substitution σ such that $[[(a,b),c],d]|_\gamma \approx_\mathsf{E} \sigma\,[(R,T),U]$ and $[([a,d],b),c]| = [[(a,b),c],d]|\,\sigma\,([R,U],T)\,|_\gamma$.

CHAPTER 4

Implementing Deep Inference in Maude

The language Maude [**CDE$^+$02, CDE$^+$03**] allows implementing term rewriting systems modulo equational theories due to its very fast matching algorithm that supports different combinations of associative commutative theories, also in the presence of units. In the previous chapter we have seen that we can consider the inference rules of the calculus of structures systems as rewrite rules corresponding to bottom-up applications of the inference rules. In the following, by exploiting this, I will present proof construction implementations of systems BV, NEL, LS, and KSg in Maude.

For this purpose, I will present systems equivalent to the above systems, where the role played by equations for exponentials (in systems NEL and LS) and units are made explicit with respect to the application of the inference rules of these systems: For the systems, NEL and LS, which admit equations for exponentials, because these equations cannot be expressed explicitly in a Maude implementation, I will convert these equations to inference rules. This way, also some redundant applications of these equations will be controlled.

Although the equations for units can be easily expressed in Maude, these equations often cause redundant matchings of the inference rules where the premise and the conclusion of the instance of the inference rules are equivalent structures, i.e., these instances are trivial instances. By redesigning the inference rules of these systems, I will make the role played by the equations for units explicit. This will result in equivalent systems where equations for exponentials (in systems with exponentials) and equations for units are redundant. By removing the equations for units from these systems, the trivial instances of the inference rules will be prevented without losing completeness.

4.1. The Maude Language

Maude [**CDE$^+$02, CDE$^+$03**] is a high level language and a high-performance system which is developed as a supporting tool for executable specifications and declarative programming in rewriting logic [**Mes92**].[1] Because rewriting logic contains a rich equational logic, namely membership equational logic [**Mes98**], Maude supports equational specification and programming. This makes it possible to express different operators modulo equational theories for associativity, commutativity and also unit, possibly different for each operator. In this section, I will give a brief introduction to language Maude. For a complete treatment the reader is referred to the Maude manual [**CDE$^+$05**], and to [**MOM02**] where many papers on rewriting logic and the language Maude are referenced.

[1] Maude can be obtained at `http://maude.cs.uiuc.edu/`.

In the language Maude, the data and the state of a system are formally specified as algebraic data types by means of an equational specification. The data types are defined by means of the keyword `sort` and subtype relations between types by means of the keyword `subsort`. The operations (the function symbols of a signature) are defined by means of the keyword `op`, by giving the types of their arguments and the type of the resulting term. Operators may have *operator attributes* which denote the associativity (`assoc`), commutativity (`comm`), idempotency (`idem`) and identity, with the corresponding term for the identity element, `id: <Term>`.

In Maude the basic units of specification and programming are called modules. There are two kinds of modules: *functional modules* and *system modules*.

4.1.1. Functional Modules. From a programming point of view, a functional module is an equational style functional program with user-definable syntax in which a number of sorts, their elements, and functions on those sorts are defined. Each functional module has a name, which is a Maude identifier. Identifiers are the basic syntactic elements, used to name modules and sorts, and to form operator names. A functional module is declared in Maude using the keywords

```
fmod <ModuleName> is <DeclarationsAndStatements>  endfm
```

As an example for a functional module let us consider the following module. This module implements the introductory example from Section 3.1 which defines natural numbers with an addition operator.

```
fmod NATURAL-NUMBERS-ADDITION is
  sort Nat .
  op e : -> Nat .
  op i : Nat -> Nat .
  op f : Nat Nat -> Nat [assoc comm] .

  vars X Y : Nat .

  eq f(X, e)    = X .
  eq f(X, i(Y)) = i(f(X,Y)) .
endfm
```

In the above module, the keyword `eq` is used to name term rewriting rules of a terminating and confluent term rewriting system. After loading the above module, we can compute the normal form of terms with respect to this functional module as follows:

```
Maude> reduce f(e, i(i(e))) .
reduce in NATURAL-NUMBERS-ADDITION : f(e, i(i(e))) .
rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
result Nat: i(i(e))
```

4.1.2. System Modules. From a programming point of view, a system module is a declarative style concurrent program with user-definable syntax. From a specification point of view, it is a rewrite theory. Again, each system module has a name, which is a Maude identifier. A system module is declared in Maude using the keywords

```
mod <ModuleName> is <DeclarationsAndStatements> endm
```

where `<DeclarationsAndStatements>` corresponds to all the declarations of submodule importations, sorts, subsorts, operators, variables, equations, rules, and so on. A module can import, or include, the definitions of another module by means of the keyword `inc` (short for `including`). A rewrite rule is defined by the keyword `rl`. In general, the (rewrite) rules specify the dynamic behavior of a distributed system. For example, the following system module defines the nondeterministic choice of a natural number from a multiset of natural numbers.

```
mod NONDETERMINISTIC-NATURAL_NUMBER is
  inc NATURAL-NUMBERS-ADDITION .
  sort Multiset .
  subsort Nat < Multiset .
  op empty : -> Multiset .
  op _,_   : Multiset Multiset -> Multiset [assoc comm id: empty] .

  var X : Nat .
  var M : Multiset .

  rl [choice] : X,M => X .
endm
```

Maude's mechanism for interleaving rules and equations then computes (the transitive closure of) the rewrite relation R/E. As an example, consider the following query where I employ the `search` command of Maude which implements breadth-first search. Below, we compute all the possible choices for a natural number from a multiset of natural numbers:

```
Maude> search i(e),e,i(i(e)),e =>* X .
search in NONDETERMINISTIC-NATURAL_NUMBER : i(e),e,e,i(i(e)) =>* X .

Solution 1 (state 1)
states: 2  rewrites: 1 in 0ms cpu (0ms real) (~ rewrites/second)
X --> e

Solution 2 (state 8)
states: 9  rewrites: 8 in 0ms cpu (0ms real) (~ rewrites/second)
X --> i(e)

Solution 3 (state 10)
states: 11  rewrites: 13 in 0ms cpu (0ms real) (~ rewrites/second)
X --> i(i(e))

No more solutions.
states: 11  rewrites: 81 in 10ms cpu (10ms real)
(8100 rewrites/second)
```

4.2. Deep Inference in Maude

In the following, I will exploit the above features, and the built-in very fast matching algorithm of Maude to implement the term rewriting systems that correspond to the systems of the calculus of structures. Besides its very simple high

level language, another important feature that makes Maude an appropriate platform for implementing systems of the calculus of structures is the availability of the search function since the 2.0 release of Maude [**CDE+03**]. This function implements breadth-first search which provides a complete search strategy for derivations and proofs. In the following, I will exploit these features for implementing systems of the calculus of structures in this language.

4.2.1. System BV in Maude. In this subsection we will see a Maude system module which implements system BV. This module presumes that the BV structures are in negation normal form. To get the negation normal form of a Σ_{BV}-term, one can employ the functional module below which implements the term rewriting system $\mathsf{R}^{\mathsf{BV}}_{Neg}$ in Definition 3.20. The function symbol $\bar{\ }$ for negation is represented by the operator `-_`. The binary function symbols $[\_,\_]$, $(\_,\_)$, and $\langle\_;\_\rangle$, respectively, are represented by the operators `[_,_]`, `{_,_}` and `<_;_>`, respectively.

```
fmod BV-NNF is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op o     : -> Unit .
  op -_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure .
  op {_,_} : Structure Structure -> Structure .
  op <_;_> : Structure Structure -> Structure .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

     eq  - o          =   o .
     eq  - [ R , T ]  =   { - R , - T } .
     eq  - { R , T }  =   [ - R , - T ] .
     eq  - < R ; T >  =   < - R ; - T > .
     eq  - - R        =   R .
endfm
```

By employing the `reduce` command of Maude, one can then compute the negation normal form of a BV structure:

```
Maude> reduce - <  [ a , - b ] ; - { c , - < d ; - o > } > .
reduce in BV-NNF : - < [a,- b] ; - {c,- < d ; - o >} > .
rewrites: 14 in 0ms cpu (0ms real) (~ rewrites/second)
result Structure: < {- a,b} ; {c,< - d ; o >} >
```

The Maude system module below implements the system RBV modulo EBV. The equations for associativity, commutativity and unit are expressed as operator attributes "`assoc`", "`comm`" and "`id : o`".

```
mod BV is
 sorts Atom Unit Structure .
 subsort Atom < Structure .
```

```
 subsort Unit < Structure .

  op o     : -> Unit .
  op -_    : Atom -> Atom [ prec 50 ].
  op [_,_] : Structure Structure -> Structure [assoc comm id: o] .
  op {_,_} : Structure Structure -> Structure [assoc comm id: o] .
  op <_;_> : Structure Structure -> Structure [assoc id: o] .

  ops a b c d e f g h i j : -> Atom .

  var R T U V : Structure .
  var A : Atom .

    rl [ai-down] : [ A , - A ]               => o .
    rl [switch]  : [ { R , T } , U ]         => { [ R , U ] , T } .
    rl [q-down]  : [ < R ; T > , < U ; V > ] => < [R,U] ; [T,V] > .
endm
```

We can then use the `search` command of Maude, which implements breadth-first search. For instance, we can search for a proof of the structure

$$[\bar{c}, \langle a; (c, \bar{b})\rangle, \langle \bar{a}; b\rangle]$$

as follows:

```
Maude> search [- c,[< a ; {c,- b} >,< - a ; b >]] =>* o .
search in BV : [- c,[< a ; {c,- b} >,< - a ; b >]] =>* o .

Solution 1 (state 2229)
states: 2230  rewrites: 196866 in 980ms cpu (1010ms real)
    (200883 rewrites/second)
empty substitution

No more solutions.
states: 2438  rewrites: 306179 in 1540ms cpu (1590ms real)
    (198817 rewrites/second)
```

It is also possible to search for derivations. For instance, we can search for a derivation of the following form:

$$\begin{array}{c} [\langle a; \bar{b}\rangle, \langle \bar{a}; b\rangle] \\ \| \mathsf{BV} \\ [\bar{c}, \langle a; (c, \bar{b})\rangle, \langle \bar{a}; b\rangle] \end{array}$$

```
Maude> search [- c,[< a ; {c,- b} >,< - a ; b >]] =>*
                                  [ < a ; - b > , < - a ; b > ] .
search in BV : [- c,[< a ; {c,- b} >,< - a ; b >]] =>*
                                      [< a ; - b >,< - a ; b >] .

Solution 1 (state 676)
states: 677  rewrites: 27969 in 130ms cpu (140ms real)
     (215146 rewrites/second)
empty substitution
```

```
No more solutions.
states: 2438  rewrites: 306179 in 1520ms cpu (1590ms real)
     (201433 rewrites/second)
```

After a successful search, we can display the derivation (proof) steps by using the command "`show path <state_number_displayed> .`":

```
Maude> show path 676 .
state 0, Structure: [- c,[< a ; {c,- b} >,< - a ; b >]]
===[ rl [< R ; T >,< U ; V >] => < [R,U] ; [V,T] > [label q-down] .
                                                                ]===>
state 30, Structure: [< a ; [- c,{c,- b}] >,< - a ; b >]
===[ rl [U,{R,T}] => {T,[R,U]} [label switch] . ]===>
state 198, Structure: [< a ; {- b,[c,- c]} >,< - a ; b >]
===[ rl [A,- A] => o [label ai-down] . ]===>
state 676, Structure: [< a ; - b >,< - a ; b >]
```

The above information displayed corresponds to the derivation

$$\mathsf{ai}{\downarrow}\,\dfrac{\mathsf{s}\,\dfrac{\mathsf{q}{\downarrow}\,\dfrac{[\langle a;\bar{b}\rangle,\langle\bar{a};b\rangle]}{[\langle a;([c,\bar{c}],\bar{b})\rangle,\langle\bar{a};b\rangle]}}{[\langle a;[\bar{c},(c,\bar{b})]\rangle,\langle\bar{a};b\rangle]}}{[\bar{c},\langle a;(c,\bar{b})\rangle,\langle\bar{a};b\rangle]}\quad .$$

It is also possible to display all the immediate instances of the rules applied to a structure by using the Maude command "`search <term> =>1 R .`".

```
Maude> search [< a ; - b >,< - a ; b >] =>1 R .
search in BV : [< a ; - b >,< - a ; b >] =>1 R .

Solution 1 (state 1)
states: 2  rewrites: 4 in 0ms cpu (0ms real) (~ rewrites/second)
R --> {< a ; - b >,< - a ; b >}

Solution 2 (state 2)
states: 3  rewrites: 11 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < - a ; [b,< a ; - b >] >

Solution 3 (state 3)
states: 4  rewrites: 12 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < - a ; < b ; < a ; - b > > >

Solution 4 (state 4)
states: 5  rewrites: 13 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < a ; [- b,< - a ; b >] >

Solution 5 (state 5)
states: 6  rewrites: 14 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < [a,- a] ; [b,- b] >

Solution 6 (state 6)
```

```
states: 7  rewrites: 15 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < [a,< - a ; b >] ; - b >

Solution 7 (state 7)
states: 8  rewrites: 16 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < a ; < - b ; < - a ; b > > >

Solution 8 (state 8)
states: 9  rewrites: 17 in 0ms cpu (0ms real) (~ rewrites/second)
R --> < [- a,< a ; - b >] ; b >

No more solutions.
states: 9  rewrites: 89 in 0ms cpu (0ms real) (~ rewrites/second)
```

4.2.2. System NEL in Maude. In the equational system ENEL, given in Definition 3.30, besides equations for associativity, commutativity and unit, which can be expressed as operator attributes in Maude, there are also equations for exponentials. These equations cannot be represented in the equational system underlying a Maude implementation of system NEL because operator attributes of Maude are allowed only for binary operators and the exponentials are unary operators. In order to get an implementation of system NEL in Maude, the role played by these equations must be captured by the inference rules. For this purpose, in the following, I will split these equations into two groups of rewrite rules: One for the left-to-right, and one for the right-to-left application of these equations. Then, I will employ the former one of these two groups at the beginning of a derivation together with the term rewriting system $\mathsf{R}^{\mathsf{NEL}}_{Neg}$ in order to obtain a normal form, which generalizes the notion of negation normal form. I will then redesign the system NEL such that this normal form will be preserved by the inference rules. This way, the equations for exponentials will become redundant in the derivations, thus they will be safely removed from the underlying equational system.

Definition 4.1. *We say that a* NEL *structure is in* exponential normal form *if it is in negation normal form, and no exponentials can be equivalently removed.*

Definition 4.2. *The term rewriting system* $\mathsf{R}^{\mathsf{NEL}}_{Exp}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{NEL}}_{Exp} = \mathsf{R}^{\mathsf{NEL}}_{Neg} \quad \cup \quad \left\{ \begin{array}{rcl} ??R & \to & ?R \\ !!R & \to & !R \\ ?\circ & \to & \circ \\ !\circ & \to & \circ \end{array} \right.$$

Proposition 4.3. *The term rewriting system* $\mathsf{R}^{\mathsf{NEL}}_{Exp}$ *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{NEL}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{NEL}}_{Exp}$ *is in exponential normal form.*

Proof. Similar to the proof of Proposition 3.24: (*i*) Take the lexicographic path order for NEL structures given in the proof of Proposition 3.24. (*ii*) Proof by analysis of the critical pairs: In addition to the critical pairs given in Proposition 3.24, we have $(!\,?\,\overline{R}, \overline{?\,R})$, $(?\,!\,\overline{R}, \overline{!\,R})$, $(!\,\bar{\circ}, \bar{\circ})$, and $(?\,\bar{\circ}, \bar{\circ})$, which are joinable. (*iii*) s being in exponential normal form and applicability of a rewrite rule of $\mathsf{R}^{\mathsf{NEL}}_{Exp}$ are contradictory. □

The exponential normal form of a NEL structure can be computed by employing the functional Maude module below, which implements the term rewriting system $\mathsf{R}^{\mathsf{NEL}}_{Exp}$. Because system NEL is a conservative extension of system BV with the exponentials of linear logic, in the modules for system NEL, the operator declarations are obtained by extending those for system BV with the operator declarations for the *of course* and *why not* structures. The function symbols ? _ and ! _, respectively, are represented by the operators ?_ and !_, respectively.

```
fmod NEL-EXP is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op o     : -> Unit .
  op -_    : Structure -> Structure .
  op ?_    : Structure -> Structure .
  op !_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure .
  op {_,_} : Structure Structure -> Structure .
  op <_;_> : Structure Structure -> Structure .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

     eq   - o          =   o .
     eq   - [ R , T ]  =   { - R , - T } .
     eq   - { R , T }  =   [ - R , - T ] .
     eq   - < R ; T >  =   < - R ; - T > .
     eq   - - R        =   R .

     eq   - ? R  =  ! - R .
     eq   - ! R  =  ? - R .

     eq   ? ? R  =  ? R .
     eq   ! ! R  =  ! R .
     eq   ? o  =  o .
     eq   ! o  =  o .
endfm
```

By employing the above module, we can compute the exponential normal form of a NEL structure as demonstrated in the following example:

```
Maude> red - [ { - a , ! ! < ? ? ? b ; - c > } , - ! < a ; b > ] .
reduce in NEL-EXP : - [{- a,! ! < ? ? ? b ; - c >},- ! < a ; b >] .
rewrites: 16 in 0ms cpu (0ms real) (~ rewrites/second)
result Structure: {[a,? < ! - b ; c >],! < a ; b >}
```

I will now redefine the equational system ENEL and system NEL such that the equations for exponentials will be removed from ENEL without damaging the completeness of the resulting system for the derivations of system NEL.

Definition 4.4. *Let* ENELe *be the equational system obtained by removing the equations for exponentials from the equational system* ENEL.

Definition 4.5. *The system in Figure 4.1 is called system* NELe. *In addition to the inference rules that are common with system* NEL, *the rules of this system are called* why not *(*?↓*),* of course *(*!↓*),* why not unit *(*?u↓*), and* of course unit *(*!u↓*). Inference rules of system* NELe *are applied on* NEL *structures, which are considered equivalent modulo the equational system* ENELe.

$$\circ\downarrow \frac{}{\circ} \qquad \mathsf{ai}\downarrow \frac{S\{\circ\}}{S[a,\bar{a}]} \qquad \mathsf{s}\,\frac{S([R,U],T)}{S[(R,T),U]} \qquad \mathsf{q}\downarrow \frac{S\langle[R,U];[T,V]\rangle}{S[\langle R;T\rangle,\langle U;V\rangle]}$$

$$\mathsf{p}\downarrow \frac{S\{![R,T]\}}{S[!R,?T]} \qquad \mathsf{w}\downarrow \frac{S\{\circ\}}{S\{?R\}} \qquad \mathsf{b}\downarrow \frac{S[?R,R]}{S\{?R\}}$$

$$?\downarrow \frac{S\{??R\}}{S\{?\,R\}} \qquad !\downarrow \frac{S\{!!R\}}{S\{!\,R\}} \qquad ?\mathsf{u}\downarrow \frac{S\{\circ\}}{S\{?\circ\}} \qquad !\mathsf{u}\downarrow \frac{S\{\circ\}}{S\{!\circ\}}$$

Figure 4.1. System NELe

Definition 4.6. *A rule* ρ *is* derivable *for a system* $\mathscr{S}$ *if for every instance* $\rho\,\dfrac{T}{R}$ *there is a derivation* $\begin{array}{l}T\\ \Vdots\mathscr{S}\\ R\end{array}$.

Definition 4.7. *Two systems* $\mathscr{S}$ *and* $\mathscr{S}'$ *are* strongly equivalent *if for every derivation* $\begin{array}{l}T\\ \Vdots\mathscr{S}\\ R\end{array}$ *there is a derivation* $\begin{array}{l}T\\ \Vdots\mathscr{S}'\\ R\end{array}$, *and vice versa.*

Proposition 4.8. *System* NEL *and system* NELe *are strongly equivalent.*

Proof. It is immediate that the rules of system NELe are derivable for system NEL, thus derivations in NELe can be rewritten as derivations in NEL. For the other direction, observe that every derivation in NEL can be equivalently written as a derivation Δ in NEL where all the structures are in exponential normal form. With induction on the length of Δ, construct the derivation Δ' in NELe: For the instances of the inference rules which do not require the application of the equations for exponentials in derivation Δ, take the same rule instance in NELe to construct Δ'. Otherwise, the following cases exhaust the other possibilities with respect to application of equations for exponentials:

- If rule $\mathsf{p}\downarrow$ is the last rule applied in Δ such that

$$\begin{array}{l}\mathsf{p}\downarrow\dfrac{S\{![T,?\,R\,]\}}{S[\,!\,T,??\,R\,]}\\ \approx\dfrac{}{S[\,!\,T,?\,R\,]}\end{array} \qquad \text{then take} \qquad \begin{array}{l}\mathsf{p}\downarrow\dfrac{S\{![T,?\,R\,]\}}{S[\,!\,T,??\,R\,]}\\ ?\downarrow\dfrac{}{S[\,!\,T,?\,R\,]}\end{array} \quad .$$

- If rule $\mathsf{p}\downarrow$ is the last rule applied in Δ such that

$$\begin{array}{c} \mathsf{p}\downarrow \dfrac{S\{![!T,R]\}}{S[!!T,?R]} \\ \approx \dfrac{}{S[!T,?R]} \end{array} \qquad \text{then take} \qquad \begin{array}{c} \mathsf{p}\downarrow \dfrac{S\{![!T,R]\}}{S[!!T,?R]} \\ !\downarrow \dfrac{}{S[!T,?R]} \end{array} \quad .$$

- If rule $\mathsf{b}\downarrow$ is the last rule applied in Δ such that

$$\begin{array}{c} \approx \dfrac{S[?R,?R]}{S[??R,?R]} \\ \mathsf{b}\downarrow \dfrac{}{S\{??R\}} \\ \approx \dfrac{}{S\{?R\}} \end{array} \qquad \text{then take} \qquad \begin{array}{c} ?\mathsf{u}\downarrow \dfrac{S[?R,?R]}{S[?\circ,?R,?R]} \\ \mathsf{w}\downarrow \dfrac{}{S[??R,?R,?R]} \\ \mathsf{b}\downarrow \dfrac{}{S[??R,?R]} \\ ?\downarrow \dfrac{}{S\{??R\}} \\ \dfrac{}{S\{?R\}} \end{array} \quad .$$

- If rule $\mathsf{ai}\downarrow$ is the last rule applied in Δ, we have the following situation. The other cases being analogous, the case for !? is as follows: (There are seven modalities in linear logic, that is, empty modality, !, ?, !?, ?!, !?! and ?!?.)

$$\begin{array}{c} \approx \dfrac{S\{\circ\}}{S\{!\circ\}} \\ \approx \dfrac{}{S\{!?\circ\}} \\ \mathsf{ai}\downarrow \dfrac{}{S\{!?[a,\bar{a}]\}} \end{array} \qquad \text{then take} \qquad \begin{array}{c} !\mathsf{u}\downarrow \dfrac{S\{\circ\}}{S\{!\circ\}} \\ ?\mathsf{u}\downarrow \dfrac{}{S\{!?\circ\}} \\ \mathsf{ai}\downarrow \dfrac{}{S\{!?[a,\bar{a}]\}} \end{array} \quad .$$

- If rule $\mathsf{w}\downarrow$ is the last rule applied in Δ, we have a situation analogous to the case for the rule $\mathsf{ai}\downarrow$ above.

□

REMARK 4.9. *In system* NELe*, the rule* $?\mathsf{u}\downarrow$ *is a redundant rule because every instance of this rule is an instance of the rule* $\mathsf{w}\downarrow$*.*

The following module implements system NELe:

```
mod NELe is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op o      : -> Unit .
  op -_     : Atom -> Atom [ prec 50 ] .
  op ?_     : Structure -> Structure [ prec 60 ] .
  op !_     : Structure -> Structure [ prec 60 ] .
  op [_,_]  : Structure Structure -> Structure [assoc comm id: o] .
  op {_,_}  : Structure Structure -> Structure [assoc comm id: o] .
  op <_;_>  : Structure Structure -> Structure [assoc id: o] .

  ops a b c d e f g h i j : -> Atom .

  var R T U V : Structure .
  var A : Atom .
```

```
  rl [ai-down]      : [ A , - A ]            => o .
  rl [switch]       : [ { R , T } , U ]      => { [ R , U ] , T } .
  rl [q-down]       : [< R ; T >,< U ; V >]  => < [R,U] ; [T,V] > .

  rl [promotion]    : [ ! R , ? T ]          => ! [ R , T ] .
  rl [weakening]    : ? R                    => o .
  rl [absorption]   : ? R                    => [ ? R , R ] .

  rl [why-not]      : ? R                    => ? ? R .
  rl [of-course]    : ! R                    => ! ! R .

  rl [wn-unit]      : ? o                    => o .
  rl [oc-unit]      : ! o                    => o .
endm
```

It is important to note that system NEL is undecidable [**Str03c**]. Also because of the rule `[absorption]`, it is not plausible to use this module for proof search without introducing a strategy which takes the application of this rule under control. However we can state the following proposition which follows immediately from Proposition 4.8.

PROPOSITION 4.10. *Let* `=>*` *denote the transitive, reflexive closure of the transition relation defined by the Maude module* `NELe`*. For* NEL *structures* R *and* T*, there is a derivation* $\begin{matrix} T \\ \| \\ R \end{matrix}_{\mathsf{NEL}}$ *if and only if* R' `=>*` T' *where* R' *and* T' *are exponential normal forms of the structures* R *and* T*.*

4.2.3. System LS in Maude. In the equational system ELS, given in Definition 3.32, besides the equations for associativity, commutativity, and unit, there are also equations for exponentials. The equations for associativity, commutativity, and units can be expressed as operator attributes in Maude. However, as in the case for system NEL, the equations for the exponentials cannot be expressed as operator attributes in a Maude implementation of system LS, because operator attributes are allowed on binary operators in Maude. In order to get an implementation of system LS in Maude, the role played by these equations must be captured by the inference rules of the deductive system. For this purpose, in the following I will apply the methods, which I used on system NEL in the previous subsection, analogously on system LS.

DEFINITION 4.11. *We say that a* LS *structure is in* exponential normal form *if it is in negation normal form, and no exponentials can be equivalently removed.*

DEFINITION 4.12. *The term rewriting system* $\mathsf{R}^{\mathsf{LS}}_{Exp}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{LS}}_{Exp} = \mathsf{R}^{\mathsf{LS}}_{Neg} \cup \left\{ \begin{array}{rcl} ??R & \to & ?R \\ !!R & \to & !R \\ ?\bot & \to & \bot \\ !1 & \to & 1 \end{array} \right.$$

PROPOSITION 4.13. *The term rewriting system* $\mathsf{R}^{\mathsf{LS}}_{Exp}$ *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{LS}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{LS}}_{Exp}$ *is in exponential normal form.*

PROOF. Similar to the proof of Proposition 3.24: (*i*) Take the lexicographic path order for LS structures given in the proof of Proposition 3.24. (*ii*) Proof by analysis of the critical pairs: In addition to the critical pairs resulting from term rewriting system $\mathsf{R}^{\mathsf{LS}}_{Neg}$, we have $(!\,?\,\overline{R}, \overline{?\,R})$, $(?\,!\,\overline{R}, \overline{!\,R})$, $(!\,\bar{\perp}, \bar{\perp})$, and $(?\,\bar{1}, \bar{1})$ that are joinable. (*iii*) s being in exponential normal form and applicability of a rewrite rule of $\mathsf{R}^{\mathsf{LS}}_{Exp}$ are contradictory. □

The exponential normal form of an LS structure can be computed by employing the functional Maude module below, which implements the term rewriting system $\mathsf{R}^{\mathsf{LS}}_{Exp}$. In the module below, the function symbol $\bar{\_}$ for negation is represented by the operator `-_`. The unary function symbols $?\_$ and $!\_$, respectively, are represented by the operators `?_` and `!_`, respectively. The binary function symbols $[\_,\_]$, $(\_,\_)$, $\{\!|\_;\_|\!\}$ and $(\!|\_;\_|\!)$, respectively, are represented by the operators `[_,_]`, `{_,_}` `[|_;_|]` and `{|_;_|}`, respectively.

```
fmod LS-EXP is

  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op 1       :  -> Unit .
  op bot     :  -> Unit .
  op 0       :  -> Unit .
  op top     :  -> Unit .

  op -_      :  Structure -> Structure .
  op ?_      :  Structure -> Structure .
  op !_      :  Structure -> Structure .

  op [_,_]   :  Structure Structure -> Structure .
  op {_,_}   :  Structure Structure -> Structure .

  op [|_,_|] :  Structure Structure -> Structure .
  op {|_,_|} :  Structure Structure -> Structure .

  ops a b c d e f g h i j  : -> Atom .

  var R T U : Structure .

     eq   - bot   =   1 .
     eq   - 1     =   bot .
     eq   - top   =   0 .
     eq   - 0     =   top .

     eq   - [ R , T ]    =   { - R , - T } .
```

```
    eq   - { R , T }    =   [ - R , - T ] .
    eq   - [| R , T |]  =   {| - R , - T |} .
    eq   - {| R , T |}  =   [| - R , - T |] .
    eq   - - R          =   R .

    eq   - ? R  =  ! - R .
    eq   - ! R  =  ? - R .

    eq   ? ? R  =  ? R .
    eq   ! ! R  =  ! R .
    eq   ? bot  =  bot .
    eq   ! 1    =  1 .
endfm
```

Similar to the module NEL-EXP, the module LS-EXP can be used to compute the exponential normal form of a LS structure:

```
Maude> red - { ? ? [| - a , b |] , ! - ! {| - a , b  |}  } .
reduce in LS-EXP : - {? ? [| - a,b |],! - ! {| - a,b |}} .
rewrites: 12 in 0ms cpu (0ms real) (~ rewrites/second)
result Structure: [! {| a,- b |},? ! {| - a,b |}]
```

I will now redefine the equational system ELS and system LS such that the equations for exponentials will be removed from ELS without damaging the completeness of the resulting systems for provable structures of system LS.

Definition 4.14. *Let* ELSe *be the equational system obtained by removing the equations for exponentials from the equational system* ELS.

Definition 4.15. *The system in Figure 4.2 is called system* LSe. *In addition to the inference rules that are common with system* LS, *the rules of this system are called* why not *(*?↓*),* of course *(*!↓*),* why not unit *(*?u↓*) and* of course unit *(*!u↓*). Inference rules of system* LSe *are applied on* LS *structures, which are considered equivalent modulo the equational system* ELSe.

Proposition 4.16. *System* LS *and system* LSe *are strongly equivalent.*

Proof. Analogous to the proof of Proposition 4.8: It is immediate that the rules of system LSe are derivable for system LS, thus derivations in LSe can be rewritten as derivations in LS. For the other direction, observe that every derivation in LS can be equivalently written as a derivation Δ in LS where all the structures are in exponential normal form. With induction on the length of Δ, construct the derivation Δ' in LSe: For the instances of the inference rules which do not require the application of the equations for exponentials in derivation Δ, take the same rule instance in LSe to construct Δ'. If the rule $\mathsf{p}\downarrow$ is the last rule applied in Δ, we have the same situation as in the Proof of Proposition 4.8. Otherwise:

$$\mathsf{1}\downarrow\frac{}{1} \qquad \mathsf{ai}\downarrow\frac{S\{1\}}{S[a,\bar{a}]} \qquad \mathsf{s}\,\frac{S([R,T],U)}{S[(R,U),T]}$$

$$\mathsf{p}\downarrow\frac{S\{![R,T]\}}{S[!R,?T]} \qquad \mathsf{w}\downarrow\frac{S\{\bot\}}{S\{?R\}} \qquad \mathsf{b}\downarrow\frac{S[?R,R]}{S\{?R\}}$$

$$\mathsf{t}\downarrow\frac{S\{0\}}{S\{R\}} \qquad \mathsf{c}\downarrow\frac{S\{R,R\}}{S\{R\}} \qquad \mathsf{d}\downarrow\frac{S\langle[R,U],[T,V]\rangle}{S[\langle R,T\rangle,\{U,V\}]}$$

$$?\downarrow\frac{S\{??R\}}{S\{?R\}} \qquad !\downarrow\frac{S\{!!R\}}{S\{!R\}} \qquad ?\mathsf{u}\downarrow\frac{S\{\bot\}}{S\{?\bot\}} \qquad !\mathsf{u}\downarrow\frac{S\{1\}}{S\{!1\}}$$

FIGURE 4.2. System LSe

- If rule $\mathsf{b}\downarrow$ is the last rule applied in Δ such that

$$\begin{array}{c}\approx\dfrac{S[?R,?R]}{S[??R,?R]}\\ \mathsf{b}\downarrow\dfrac{}{S\{??R\}}\\ \approx\dfrac{}{S\{?R\}}\end{array} \qquad \text{then take} \qquad \begin{array}{c}?\mathsf{u}\downarrow\dfrac{S[?R,?R]}{S[?\bot,?R,?R]}\\ \mathsf{w}\downarrow\dfrac{}{S[??R,?R,?R]}\\ \mathsf{b}\downarrow\dfrac{}{S[??R,?R]}\\ ?\downarrow\dfrac{}{S\{?R\}}\end{array} .$$

- If rule $\mathsf{ai}\downarrow$ is the last rule applied in Δ, we have the following situation.

$$\begin{array}{c}\approx\dfrac{S\{1\}}{S\{!1\}}\\ \mathsf{ai}\downarrow\dfrac{}{S\{![a,\bar{a}]\}}\end{array} \qquad \text{then take} \qquad \begin{array}{c}!\mathsf{u}\downarrow\dfrac{S\{1\}}{S\{!1\}}\\ \mathsf{ai}\downarrow\dfrac{}{S\{![a,\bar{a}]\}}\end{array} .$$

□

REMARK 4.17. *In system* LSe*, the rule* $?\mathsf{u}\downarrow$ *is a redundant rule because every instance of this rule is an instance of the rule* $\mathsf{w}\downarrow$*.*

The following module implements system LSe:

```
mod LSe is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op 1       : -> Unit .
  op bot     : -> Unit .
  op 0       : -> Unit .
  op top     : -> Unit .
```

```
  op -_      : Atom -> Atom [ prec 50 ] .
  op ?_      : Structure -> Structure [ prec 60 ] .
  op !_      : Structure -> Structure [ prec 60 ] .
  op [_,_]   : Structure Structure -> Structure [assoc comm id: bot] .
  op {_,_}   : Structure Structure -> Structure [assoc comm id: 1] .
  op [|_,_|] : Structure Structure -> Structure [assoc comm id: 0] .
  op {|_,_|} : Structure Structure -> Structure [assoc comm id: top] .

  ops a b c d e f g h i j : -> Atom .

  var R T U V : Structure .    var A : Atom .

    rl [atomic-int.]  :  [ A , - A ]        =>  1 .
    rl [switch]       :  [ { R , T } , U ]  =>  { [ R , U ] , T } .

    rl [additive]     :  [ {| R , T |} , [| U , V |] ]
                                  =>  {| [ R , U ] , [ T , V ]  |} .

    rl [thinning]     :  R                   =>  0 .
    rl [contraction]  :  R                   =>  [| R , R |] .

    rl [promotion]    :  [ ! R , ? T ]       =>  ! [ R , T ] .
    rl [weakening]    :  ? R                 =>  bot .
    rl [absorption]   :  ? R                 =>  [ ? R , R ] .

    rl [of-course]    :  ? R                 => ? ? R .
    rl [why-not]      :  ! R                 => ! ! R .
    rl [?-unit]       :  ? bot               =>  bot .
    rl [!-unit]       :  ! 1                 =>  1 .

    rl [one-with-l]   :  {| 1 , 1 |}         =>  1 .
    rl [one-with-r]   :  1                   => {| 1 , 1 |} .
    rl [bot-with-l]   :  [| bot , bot |]     =>  bot .
    rl [bot-with-r]   :  bot                 => [| bot , bot |] .
endm
```

The equations of system ELSe include the equations $\{1, 1\} \approx 1$ and $[\bot, \bot] \approx \bot$. These equations cannot be captured by the operator attributes for units in a Maude implementation, because Maude operator attributes do not allow such a usage. Thus, in order to implement system LSe as a Maude module, the role played by these equations must be simulated by means of Maude rules. The last four rules in the above module serve this purpose.

As it is the case for system NEL, linear logic is known to be undecidable [**LMSS90**]. Also for this module some strategy, that takes the rules `[absorption]` and `[contraction]` under control, is necessary for proof search applications. However, we can state the following proposition which follows immediately from Proposition 4.16.

PROPOSITION 4.18. *Let* `=>*` *denote the transitive, reflexive closure of the transition relation defined by the Maude module* `LSe`. *For* LS *structures* R *and* T, *there*

is a derivation $\begin{matrix} T \\ \| \mathsf{LS} \\ R \end{matrix}$ *if and only if* R' `=>*` T' *where* R' *and* T' *are exponential normal forms of the structures* R *and* T.

4.2.4. System KSg in Maude. To get the negation normal form of a Σ_{KSg}-term, one can employ the functional module below, which implements the term rewriting system $\mathsf{R}^{\mathsf{KSg}}_{Neg}$ in Definition 3.23.

```
fmod KSg-NNF is
  sorts Unit Atom Structure .
  subsort Unit < Structure .
  subsort Atom < Structure .

  op tt    : -> Unit .
  op ff    : -> Unit .
  op -_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure .
  op {_,_} : Structure Structure -> Structure .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

     eq  - tt         =  ff .
     eq  - ff         =  tt .
     eq  - [ R , T ]  =  { - R , - T } .
     eq  - { R , T }  =  [ - R , - T ] .
     eq  - - R        =  R .
endfm
```

This module can be used analogously as the module `BV-NNF` to obtain the negation normal forms of the KSg structures.

The Maude system module below implements the system RKSg modulo EKSg.

```
mod KSg is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op tt    : -> Unit .
  op ff    : -> Unit .
  op -_    : Atom -> Atom [ prec 50 ].
  op [_,_] : Structure Structure -> Structure [assoc comm id: ff] .
  op {_,_} : Structure Structure -> Structure [assoc comm id: tt] .

  ops a b c d e f g h i j : -> Atom .

   var R T U  V : Structure .
   var A : Atom .

     rl [atomic-int.] :  [ A , - A ]        => tt .
```

```
    rl [switch]      :  [ { R , T } , U ]  => { [ R , U ] , T } .
    rl [weakening]   :  R                  => ff .
    rl [contraction] :  R                  => [ R , R ] .

    rl [tt-dis]      :  tt   =>   [ tt , tt ] .
    rl [ff-con]      :  ff   =>   { ff , ff } .
endm
```

In the above module, the last two rules are added to the module in order to model the right to left application of the equations $[\mathtt{t},\mathtt{t}] \approx \mathtt{t}$ and $(\mathtt{f},\mathtt{f}) \approx \mathtt{f}$ in arbitrary KSg derivations.

Remark 4.19. *In an implementation of system* KSg *in Maude, one might consider to add the inference rules*

$$\frac{S\{\mathtt{t}\}}{S[\mathtt{t},\mathtt{t}]} \qquad \textit{and} \qquad \frac{S\{\mathtt{f}\}}{S(\mathtt{f},\mathtt{f})}$$

to capture the role played by the equations in the equational system EKSg*, because the equations corresponding to these rules are not captured by the operator attributes for unit in a Maude implementation. However, it can be easily observed that both of these rules are instances of the rule* $\mathsf{w}\!\downarrow$*:*

$$\begin{array}{c} \approx \dfrac{S\{\mathtt{t}\}}{S[\mathtt{t},\mathtt{f}]} \\ \mathsf{w}\!\downarrow \dfrac{}{S[\mathtt{t},\mathtt{t}]} \end{array} \qquad\qquad \mathsf{w}\!\downarrow \frac{S\{\mathtt{f}\}}{S(\mathtt{f},\mathtt{f})}$$

Although classical logic is decidable (coNP-complete), due to the `[contraction]` rule, which copies arbitrary structures in a proof search episode, it is not plausible to use the above module for proof search purposes. In Subsection 4.3.4, I will present a contraction-free system for classical logic in the calculus of structures which can be used for proof search purposes. However, we can state the following proposition which follows immediately from Remark 4.19 and Proposition 3.35.

Proposition 4.20. *Let* `=>*` *denote the transitive, reflexive closure of the transition relation defined by the Maude module* `KSg`*. For* KSg *structures* R *and* T*, there is a derivation* $\begin{array}{c} T \\ \| \\ R \end{array}\!\!\mathsf{KSg}$ *if and only if* R' `=>*` T' *where* R' *and* T' *are negation normal forms of the structures* R *and* T*.*

4.3. Removing the Equations for Unit

In the above implementations the structures must be matched modulo an equational system, which consists of equations for associativity, commutativity, and units for different logical operators. In the modules that I presented in the previous sections, these equational systems are expressed as operator attributes. However, at a closer inspection of the inference rules of these systems, it is easy to see that the equations for the units often cause trivial instances of the inference rules. For instance, consider the following instances of the switch rule in the systems BV, LS

and KSg, respectively:

$$\approx\dfrac{[R,T]}{\mathsf{s}\dfrac{([R,T],\circ)}{\approx\dfrac{[(R,\circ),T]}{[R,T]}}} \qquad \approx\dfrac{[R,T]}{\mathsf{s}\dfrac{([R,T],1)}{\approx\dfrac{[(R,1),T]}{[R,T]}}} \qquad \approx\dfrac{[R,T]}{\mathsf{s}\dfrac{([R,T],\mathtt{tt})}{\approx\dfrac{[(R,\mathtt{tt}),T]}{[R,T]}}}$$

$$\approx\dfrac{(R,T)}{\mathsf{s}\dfrac{([R,\circ],T)}{\approx\dfrac{[(R,T),\circ]}{(R,T)}}} \qquad \approx\dfrac{(R,T)}{\mathsf{s}\dfrac{([R,\bot],T)}{\approx\dfrac{[(R,T),\bot]}{(R,T)}}} \qquad \approx\dfrac{(R,T)}{\mathsf{s}\dfrac{([R,\mathtt{ff}],T)}{\approx\dfrac{[(R,T),\mathtt{ff}]}{(R,T)}}}$$

One can observe a similar behavior in the seq rule of systems BV and NEL:

$$\approx\dfrac{\langle R;T\rangle}{\mathsf{q}\downarrow\dfrac{\langle[R,\circ];[\circ,T]\rangle}{\approx\dfrac{[\langle R;\circ\rangle,\langle\circ;T\rangle]}{\langle R;T\rangle}}} \qquad \approx\dfrac{[R,T]}{\mathsf{q}\downarrow\dfrac{\langle[R,T];[\circ,\circ]\rangle}{\approx\dfrac{[\langle R;\circ\rangle,\langle T;\circ\rangle]}{[R,T]}}}$$

In a proof search episode, such trivial instances of the inference rules cause redundant branchings in the search space. In the following, I will present systems equivalent to the above mentioned systems, where the rule applications with respect to the equations for the units are made explicit by redesigning the inference rules of these systems. This way, the equations for units will be safely removed from the underlying equational system without damaging the completeness of these systems. I will then demonstrate, on Maude modules, that indeed the resulting systems perform much better in automated proof search.

4.3.1. Equations for Unit in System BV. At a first step for removing the equations for the unit from the underlying equational theory of system BV, we extend the term rewriting system $\mathsf{R}^{\mathsf{BV}}_{Neg}$ to obtain unit normal forms of the structures.

Definition 4.21. *Let* EBVu *be the equational system obtained by removing the equations for the unit from the equational system* EBV.

Definition 4.22. *The term rewriting system* $\mathsf{R}^{\mathsf{BV}}_{Unit}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{BV}}_{Unit} = \mathsf{R}^{\mathsf{BV}}_{Neg} \cup \left\{\begin{array}{llll} [\,R\,,\circ\,] & \to & R & [\,\circ\,,R\,] \to R \\ (\,R\,,\circ\,) & \to & R & (\,\circ\,,R\,) \to R \\ \langle\,R\,;\circ\,\rangle & \to & R & \langle\,\circ\,;R\,\rangle \to R \end{array}\right.$$

Proposition 4.23. *The term rewriting system* $\mathsf{R}^{\mathsf{BV}}_{Unit}$ *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{BV}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{BV}}_{Unit}$ *is in unit normal form.*

Proof. Similar to the proof of Proposition 3.24: (*i*) Take the lexicographic path order for BV structures given in the proof of Proposition 3.24. (*ii*) Proof by analysis of the critical pairs: In addition to the critical pairs resulting from term rewriting system $\mathsf{R}^{\mathsf{BV}}_{Neg}$, we have $(\,(\bar{R},\bar{\circ}),\bar{R}\,)$, $(\,\bar{R},(\bar{R},\bar{\circ})\,)$, $(\,[\bar{R},\bar{\circ}],\bar{R}\,)$, $(\,\bar{R},[\bar{R},\bar{\circ}]\,)$,

$(\langle \bar{R}; \bar{\circ}\rangle, \bar{R})$, and $(\bar{R}, \langle \bar{R}; \bar{\circ}\rangle)$ that are joinable. (*iii*) s being in unit normal form and applicability of a rewrite rule of $\mathsf{R}^{\mathsf{BV}}_{Unit}$ are contradictory.

□

The following functional Maude module implements the term rewriting system $\mathsf{R}^{\mathsf{BV}}_{Unit}$.

```
fmod BV-UNF is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op o     : -> Unit .
  op -_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure .
  op {_,_} : Structure Structure -> Structure .
  op <_;_> : Structure Structure -> Structure .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

    eq   - o   = o .
    eq   - [ R , T ] = { - R , - T } .
    eq   - { R , T } = [ - R , - T ] .
    eq   - < R ; T > = < - R ; - T > .
    eq   - - R  = R .

    eq   [ R , o ] = R .      eq   [ o , R ] = R .
    eq   { R , o } = R .      eq   { o , R } = R .
    eq   < R ; o > = R .      eq   < o ; R > = R .
endfm
```

Definition 4.24. *The system shown in Figure 4.3 is called system* BVn. *The rules of this system are called* unit *(*$\circ\downarrow$*),* atomic interaction *(*$\mathsf{ai}\downarrow$*),* switch 1 *(*s_1*),* switch 2 *(*s_2*),* seq 1 *(*$\mathsf{q}_1\downarrow$*),* seq 2 *(*$\mathsf{q}_2\downarrow$*),* seq 3 *(*$\mathsf{q}_3\downarrow$*),* seq 4 *(*$\mathsf{q}_4\downarrow$*),* unit 1 *(*$\mathsf{u}_1\downarrow$*),* unit 2 *(*$\mathsf{u}_2\downarrow$*),* unit 3 *(*$\mathsf{u}_3\downarrow$*), and* unit 4 *(*$\mathsf{u}_4\downarrow$*). Inference rules of system* BVn *are applied on* BV *structures, which are considered equivalent modulo the equational system* EBVu.

One can observe the similarity between the switch rule and seq rule, in particular the rules $\mathsf{q}_3\downarrow$ and $\mathsf{q}_4\downarrow$. In fact, Retoré gives rules similar to the rules $\mathsf{q}_1\downarrow$, $\mathsf{q}_2\downarrow$, $\mathsf{q}_3\downarrow$, $\mathsf{q}_4\downarrow$, for the Pomset Logic in [**Ret97**], which is conjectured to be equivalent to BV in [**Str03a**]. Although Retoré does not provide a cut elimination proof for his system, cut elimination for the systems, where the rule $\mathsf{q}\downarrow$ is replaced with these rules, follows from Theorem 2.19 and the results that I will present in this section.

Lemma 4.25. *The rules* $\mathsf{q}_1\downarrow$, $\mathsf{q}_2\downarrow$, $\mathsf{q}_3\downarrow$, *and* $\mathsf{q}_4\downarrow$ *are derivable for* $\{\mathsf{q}\downarrow\}$. *The rules* s_1 *and* s_2 *are derivable for* $\{\mathsf{s}\}$.

Proof. We do the following case analysis:

- For the rule $\mathsf{q}_1\downarrow$ take the rule $\mathsf{q}\downarrow$.

$$\circ\downarrow \frac{}{\circ} \qquad \mathsf{ai}\downarrow \frac{S\{\circ\}}{S[a,\bar{a}]} \qquad \mathsf{s}_1 \frac{S([R,U],T)}{S[(R,T),U]} \qquad \mathsf{s}_2 \frac{S(R,T)}{S[R,T]}$$

$$\mathsf{q}_1\downarrow \frac{S\langle[R,U];[T,V]\rangle}{S[\langle R;T\rangle,\langle U;V\rangle]} \qquad \mathsf{q}_2\downarrow \frac{S\langle R;T\rangle}{S[R,T]} \qquad \mathsf{q}_3\downarrow \frac{S\langle[R,U];T\rangle}{S[\langle R;T\rangle,U]} \qquad \mathsf{q}_4\downarrow \frac{S\langle R;[T,U]\rangle}{S[\langle R;T\rangle,U]}$$

$$\mathsf{u}_1\downarrow \frac{S\{R\}}{S[R,\circ]} \qquad \mathsf{u}_2\downarrow \frac{S\{R\}}{S(R,\circ)} \qquad \mathsf{u}_3\downarrow \frac{S\{R\}}{S\langle R;\circ\rangle} \qquad \mathsf{u}_4\downarrow \frac{S\{R\}}{S\langle\circ;R\rangle}$$

FIGURE 4.3. System BVn

- For the rule $\mathsf{q}_2\downarrow$, $\mathsf{q}_3\downarrow$, $\mathsf{q}_4\downarrow$, respectively, take the following derivations, respectively:

$$\begin{array}{c}\approx\dfrac{\langle R;T\rangle}{\langle[R,\circ];[\circ,T]\rangle}\\ \mathsf{q}\downarrow\dfrac{}{[\langle R;\circ\rangle,\langle\circ;T\rangle]}\\ \approx\dfrac{}{[R,T]}\end{array} \qquad \begin{array}{c}\approx\dfrac{\langle[R,U];T\rangle}{\langle[R,U];[T,\circ]\rangle}\\ \mathsf{q}\downarrow\dfrac{}{[\langle R;T\rangle,\langle U;\circ\rangle]}\\ \approx\dfrac{}{[\langle R;T\rangle,U]}\end{array} \qquad \begin{array}{c}\approx\dfrac{\langle R;[T,U]\rangle}{\langle[R,\circ];[T,U]\rangle}\\ \mathsf{q}\downarrow\dfrac{}{[\langle R;T\rangle,\langle\circ;U\rangle]}\\ \approx\dfrac{}{[\langle R;T\rangle,U]}\end{array}$$

- For the rule s_1 take the rule s.
- For the rule s_2 take the following derivation:

$$\begin{array}{c}\approx\dfrac{(R,T)}{([\circ,T],R)}\\ \mathsf{s}\dfrac{}{[(\circ,R),T]}\\ \approx\dfrac{}{[R,T]}\end{array} \quad . \qquad \square$$

PROPOSITION 4.26. *Every* BV *structure in negation normal form can be transformed to a structure in unit normal form by applying the rules* $\{\mathsf{u}_1\downarrow,\mathsf{u}_2\downarrow,\mathsf{u}_3\downarrow,\mathsf{u}_4\downarrow\}$ *in Figure 4.3 bottom-up.*

PROOF. Because the bottom-up application of the rules $\{\mathsf{u}_1\downarrow,\mathsf{u}_2\downarrow,\mathsf{u}_3\downarrow,\mathsf{u}_4\downarrow\}$ to a BV structure corresponds to the application of the rewriting rules in $\mathsf{R}^{\mathsf{BV}}_{Unit} \setminus \mathsf{R}^{\mathsf{BV}}_{Neg}$ in Definition 4.22, the result follows immediately from Proposition 4.23. $\square$

LEMMA 4.27. *For every derivation* $\begin{array}{c}W\\ \Delta\,\|\,\mathsf{BV}\\ Q\end{array}$ *there exists a derivation* $\begin{array}{c}W'\\ \Delta'\,\|\,\mathsf{BVn}\\ Q\end{array}$ *where* W' *is a unit normal form of the structure* W*.*

PROOF. Observe that every derivation Δ in BV can be equivalently written as a derivation Δ where all the structures are in unit normal form. From Proposition 4.26 we get a normal form Q' of Q while going up in a derivation. With induction on the length of Δ we will construct the derivation

$$\begin{array}{c}W'\\ \Delta'\,\|\,\mathsf{BVn}\\ Q'\end{array} .$$

- If Δ is $\circ\downarrow\dfrac{}{\circ}$ then take $\Delta' = \Delta$.
- If, for an atom a, $\mathsf{ai}\downarrow\dfrac{S\{\circ\}}{S[a,\bar{a}]}$ is the last rule applied in Δ, then by Proposition 4.26 and by the induction hypothesis there is a derivation $\begin{array}{l} W' \\ \Vert \mathsf{BVn} \\ P \end{array}$ where P is a normal form of $S\{\circ\}$. The following cases exhaust the possibilities For some structure T and a struture context $S'\{\ \}$:
  - If $S\{\ \} = S'[T, \{\ \}]$ then take the derivation
$$\begin{array}{c} \mathsf{u}_1\downarrow\dfrac{S'\{T\}}{S'[\,T\,,\,\circ\,]} \\ \mathsf{ai}\downarrow\dfrac{}{S'[T\,,\,[a,\bar{a}]]} \end{array} \quad .$$
  - If $S\{\ \} = S'(T, \{\ \})$ then take the derivation
$$\begin{array}{c} \mathsf{u}_2\downarrow\dfrac{S'\{T\}}{S'(\,T\,,\,\circ\,)} \\ \mathsf{ai}\downarrow\dfrac{}{S'(T\,,\,[a,\bar{a}])} \end{array} \quad .$$
  - If $S\{\ \} = S'\langle T; \{\ \}\rangle$ then take the derivation
$$\begin{array}{c} \mathsf{u}_3\downarrow\dfrac{S'\{T\}}{S'\langle\,T\,;\,\circ\,\rangle} \\ \mathsf{ai}\downarrow\dfrac{}{S'\langle T\,;\,[a,\bar{a}]\rangle} \end{array} \quad .$$
  - If $S\{\ \} = S'\langle\{\ \}; T\rangle$ then take the following derivation.
$$\begin{array}{c} \mathsf{u}_4\downarrow\dfrac{S'\{T\}}{S'\langle\,\circ\,;\,T\,\rangle} \\ \mathsf{ai}\downarrow\dfrac{}{S'\langle[a,\bar{a}];T\,\rangle} \end{array} \quad .$$
- If $\mathsf{s}\dfrac{P}{Q}$ is the last rule applied in Δ where $Q = S[(R,T),U]$ for some context S and structures R, T and U, then by induction hypothesis there is a derivation $\begin{array}{l} W' \\ \Vert \mathsf{BVn} \\ P \end{array}$. We do case analysis with respect to the unit. The following cases exhaust the possibilities:
  - If $R \neq \circ$, $T \neq \circ$, and $U \neq \circ$, then apply the rule s_1 to Q'.
  - If $R = \circ$, $T \neq \circ$, and $U \neq \circ$, then $Q' = S'[T,U]$ where S' is a normal form of context S. Apply the rule s_2 to Q'.
  - Other 6 cases, where $T = \circ$ or $U = \circ$, are trivial instances of the s rule. Take $P = Q'$.

- If $\mathsf{q}{\downarrow}\dfrac{P}{Q}$ is the last rule applied in Δ where $Q = S[\langle R;T\rangle,\langle U;V\rangle]$ for some context S and structures R, T, U and V, then by induction hypothesis there is a derivation $\begin{array}{c} W' \\ \| \\ P \end{array}_{\mathsf{BVn}}$. We do case analysis with respect to the unit. The following cases exhaust the possibilities:

  - If $R \neq \circ$, $T \neq \circ$, $U \neq \circ$, and $V \neq \circ$, then apply the rule $\mathsf{q}_1{\downarrow}$ to Q'.
  - If $R = \circ$, $T \neq \circ$, $U \neq \circ$, and $V \neq \circ$, then $Q' = S'[T,\langle U;V\rangle]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_4{\downarrow}$ to Q'.
  - If $R \neq \circ$, $T = \circ$, $U \neq \circ$, and $V \neq \circ$, then $Q' = S'[R,\langle U;V\rangle]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_3{\downarrow}$ to Q'.
  - If $R \neq \circ$, $T \neq \circ$, $U = \circ$, and $V \neq \circ$, then $Q' = S'[[R;T],V]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_4{\downarrow}$ to Q'.
  - If $R \neq \circ$, $T \neq \circ$, $U \neq \circ$, and $V = \circ$, then $Q' = S'[\langle R;T\rangle,U]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_3{\downarrow}$ to Q'.
  - If $R \neq \circ$, $T = \circ$, $U = \circ$, and $V \neq \circ$, then $Q' = S'[R,V]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_2{\downarrow}$ to Q'.
  - If $R = \circ$, $T \neq \circ$, $U \neq \circ$, and $V = \circ$, then $Q' = S'[T,U]$ where S' is a normal form of context S. Apply the rule $\mathsf{q}_2{\downarrow}$ to Q'.
  - The 4 cases where $R = \circ$, and $T = \circ$ are trivial instances of the $\mathsf{q}{\downarrow}$ rule. Take $P = Q'$.
  - The 2 cases where $R = \circ$, $T \neq \circ$, and $U = \circ$ are trivial instances of the $\mathsf{q}{\downarrow}$ rule. Take $P = Q'$.
  - The 3 cases where $R = \circ$, $V =\neq \circ$, and either $T \neq \circ$, and $U \neq \circ$, or $T \neq \circ$, and $U = \circ$, or $T = \circ$, and $U \neq \circ$ are trivial instances of the $\mathsf{q}{\downarrow}$ rule. Take $P = Q'$.

□

THEOREM 4.28. *System* BV *and system* BVn *are strongly equivalent.*

PROOF. From Lemma 4.25 it follows that the derivations in BVn are also derivations in BV. Derivations in BV are translated to derivations in BVn by Lemma 4.27. □

COROLLARY 4.29. *The systems* $\{\,\mathsf{q}{\downarrow}\,\}$ *and* $\{\,\mathsf{q}_1{\downarrow},\mathsf{q}_2{\downarrow},\mathsf{q}_3{\downarrow},\mathsf{q}_4{\downarrow}\,\}$ *are equivalent.*

PROOF. The result follows immediately from Lemma 4.25 and the cases for the rule $\mathsf{q}{\downarrow}$ of the proof of Lemma 4.27. □

The below Maude system module implements system BVn:

```
mod BVn is
 sorts Atom Unit Structure .
 subsort Atom < Structure .
 subsort Unit < Structure .

  op o       : -> Unit .
  op -_      : Atom -> Atom [ prec 50 ].
  op [_,_]   : Structure Structure -> Structure [assoc comm] .
  op {_,_}   : Structure Structure -> Structure [assoc comm] .
  op <_;_>   : Structure Structure -> Structure [assoc] .
  ops a b c d e f g h i j : -> Atom .

 var R T U V : Structure .
 var A : Atom .

 rl [ai] : [ A , - A ]  => o .

 rl [switch-1] :  [ { R , T } , U ]    => { [ R , U ] , T } .
 rl [switch-2] :  [ R , T ]            => { R, T } .

 rl [seq-1] : [ < R ; T > , < U ; V > ] => < [R,U] ; [T,V] > .
 rl [seq-2] : [ R , T ]                  => < R ; T > .
 rl [seq-3] : [ < R ; T > , U ]          => < [ R , U ] ; T > .
 rl [seq-4] : [ < R ; T > , U ]          => < R ; [ T , U ] > .

 rl [unit-1] :  [ R , o ]   =>  R .
 rl [unit-2] :  { R , o }   =>  R .
 rl [unit-3] :  < R ; o >   =>  R .
 rl [unit-4] :  < o ; R >   =>  R .
endm
```

REMARK 4.30. *From the point of view of bottom-up proof search, rule* s_2 *is a redundant rule because the structures in a copar structure cannot interact with each other with a bottom-up application of an inference rule.*[2] *Hence, this rule does not play any role from the point of view of provability of* BV *structures because an application of this rule disables the interaction between two structures in a proof search episode. However, in order to preserve completeness for arbitrary derivations, I included this rule in system* BVn.

By resorting to the observations that are made while proving Lemma 4.27, we will now see that it is possible to remove the unit $\circ$ completely from the language of BV structures:

DEFINITION 4.31. *The system in Figure 4.4 is called system* BVu*, or unit-free system* BV*. In addition to the inference rules that are common with system* BVn*, the rules of this system are called* axiom *(*ax*),* atomic interaction 1 *(*$\mathsf{ai}_1{\downarrow}$*),* atomic

[2]In Chapter 5, I will present a system equivalent to system BV, where the nondeterminism in proof search is reduced by imposing a simple restriction on the application of the switch rule. This system, called BVsl, allows to see this observation explicitly.

interaction 2 *(*$\mathsf{ai}_2{\downarrow}$*)*, atomic interaction 3 *(*$\mathsf{ai}_3{\downarrow}$*), and* atomic interaction 4 *(*$\mathsf{ai}_4{\downarrow}$*). Inference rules of system* BVu *are applied on* BV *structures, which are considered equivalent modulo the equational system* EBVu.

The inference rules $\mathsf{ai}_1{\downarrow}, \mathsf{ai}_2{\downarrow}, \mathsf{ai}_3{\downarrow}$, and $\mathsf{ai}_4{\downarrow}$ of system BVu are obtained by merging each of the rules $\mathsf{u}_1{\downarrow}, \mathsf{u}_2{\downarrow}, \mathsf{u}_3{\downarrow}$, and $\mathsf{u}_4{\downarrow}$ in Figure 4.3 with the rule $\mathsf{ai}{\downarrow}$.

$$\mathsf{ax}\,\frac{}{[a,\bar{a}]} \qquad \mathsf{s}_1\,\frac{S([R,W],T)}{S[(R,T),W]}$$

$$\mathsf{ai}_1{\downarrow}\,\frac{S\{R\}}{S[R,[a,\bar{a}]]} \qquad \mathsf{ai}_2{\downarrow}\,\frac{S\{R\}}{S(R,[a,\bar{a}])} \qquad \mathsf{ai}_3{\downarrow}\,\frac{S\{R\}}{S\langle R;[a,\bar{a}]\rangle} \qquad \mathsf{ai}_4{\downarrow}\,\frac{S\{R\}}{S\langle [a,\bar{a}];R\rangle}$$

$$\mathsf{q}_1{\downarrow}\,\frac{S\langle [R,U];[T,V]\rangle}{S[\langle R;T\rangle,\langle U;V\rangle]} \qquad \mathsf{q}_2{\downarrow}\,\frac{S\langle R;T\rangle}{S[R,T]} \qquad \mathsf{q}_3{\downarrow}\,\frac{S\langle [R,W];T\rangle}{S[\langle R;T\rangle,W]} \qquad \mathsf{q}_4{\downarrow}\,\frac{S\langle R;[T,W]\rangle}{S[\langle R;T\rangle,W]}$$

FIGURE 4.4. System BVu

DEFINITION 4.32. *Two systems* $\mathscr{S}$ *and* $\mathscr{S}'$ *are* (weakly) equivalent *if for every proof* $\prod_R^{\mathscr{S}}$ *there is a proof* $\prod_R^{\mathscr{S}'}$, *and vice versa.*

COROLLARY 4.33. *System* BV *and system* $\mathsf{BVu} \cup \{\circ{\downarrow}\}$ *are equivalent.*

PROOF. The rules $\mathsf{ai}_1{\downarrow}, \mathsf{ai}_2{\downarrow}, \mathsf{ai}_3{\downarrow}, \mathsf{ai}_4{\downarrow}$, and ax are derivable for system BVn. Other rules being similar, let us see this for the rules $\mathsf{ai}_1{\downarrow}$, and ax:

$$\mathsf{u}_1{\downarrow}\,\frac{S\{R\}}{S[R,\circ]}\qquad \mathsf{ai}{\downarrow}\,\frac{S[R,\circ]}{S[R,[a,\bar{a}]]} \qquad\qquad \circ{\downarrow}\,\frac{}{\circ}\qquad \mathsf{ai}{\downarrow}\,\frac{\circ}{[a,\bar{a}]}$$

The other direction follows from the proof of Lemma 4.27 and Remark 4.30. □

The following Maude system module implements system BVn:

```
mod BVu is
 sorts Atom Structure .
 subsort Atom < Structure .

  op -_      : Atom -> Atom [ prec 50 ].
  op [_,_]   : Structure Structure -> Structure [assoc comm] .
  op {_,_}   : Structure Structure -> Structure [assoc comm] .
  op <_;_>   : Structure Structure -> Structure [assoc] .

  ops a b c d e f g h i j : -> Atom .

 var R T U V : Structure .
 var A : Atom .
```

```
 rl [ai-1]  :  [ R , [ A , - A ] ]  =>  R .
 rl [ai-2]  :  { R , [ A , - A ] }  =>  R .
 rl [ai-3]  :  < R ; [ A , - A ] >  =>  R .
 rl [ai-4]  :  < [ A , - A ] ; R >  =>  R .

 rl [switch-1] : [ { R , T } , U ]  => { [ R , U ] , T } .

 rl [seq-1] : [ < R ; T > , < U ; V > ] => < [R,U] ; [T,V] > .
 rl [seq-2] : [ R , T ]                 => < R ; T > .
 rl [seq-3] : [ < R ; T > , U ]         => < [ R , U ] ; T > .
 rl [seq-4] : [ < R ; T > , U ]         => < R ; [ T , U ] > .
endm
```

Similar to the other modules presented so far for system BV, this module can be used for proof search. However, because the axiom scheme of this system is different than the other systems the search queries in Maude are slightly different:

```
Maude> search [- c,[< a ; {c,- b} >,< - a ; b >]] =>* [A,- A] .
search in BVu : [- c,[< a ; {c,- b} >,< - a ; b >]] =>* [A,- A] .

Solution 1 (state 521)
states: 522  rewrites: 1416 in 10ms cpu (10ms real)
        (141600 rewrites/second)
A --> b

Solution 2 (state 544)
states: 545  rewrites: 1448 in 10ms cpu (10ms real)
        (144800 rewrites/second)
A --> c

Solution 3 (state 984)
states: 985  rewrites: 2961 in 30ms cpu (30ms real)
        (98700 rewrites/second)
A --> a

No more solutions.
states: 1243  rewrites: 4691 in 50ms cpu (50ms real)
        (93820 rewrites/second)
```

I will now give a comparison of the systems BV, BVn and BVu with respect to the implementations of these systems. The tables below show the cpu time spent and the number of rewrites taken while proving the respective BV structures in modules for systems BV, BVn and BVu. All the experiments below are performed on an Intel Pentium 1400 MHz Processor.

Consider the following structure that I took from [**Bru02**], which corresponds to a process expression in a process algebra, called $\mathsf{PA}_{\mathsf{BV}}$, which is a fragment of the process algebra CCS [**Mil89**]. This example is particularly interesting, because there is a strict correspondence between the process algebra $\mathsf{PA}_{\mathsf{BV}}$ and system BV.

$$[a, \langle a; [c, \bar{a}]\rangle, \langle \bar{a}; \bar{c}\rangle]$$

The following table provides a performance comparison in search for a proof of this structure in systems BV, BVn and BVu:

System	finds a proof		search terminates	
	in # millisec.	after # rewrites	in # millisec.	after # rewrites
BV	1370	281669	5530	1100629
BVn	500	59734	560	65273
BVu	0	581	140	15244

When we search for a proof of a similar query which involves also copar structures we get the following results:

$$[\bar{c}, \langle a; (c, \bar{b})\rangle, \langle \bar{a}; \bar{b}\rangle]$$

System	finds a proof		search terminates	
	in # millisec.	after # rewrites	in # millisec.	after # rewrites
BV	950	196866	1490	306179
BVn	120	12610	120	12720
BVu	10	1416	60	4691

Table 4.1 gives a representative performance comparison of these systems on the below proof search queries:

(1) $[\langle a; [b, c]\rangle, \langle [\bar{a}, \bar{b}]; \bar{c}\rangle]$
(2) $[a, b, (\bar{a}, \bar{c}), (\bar{b}, c)]$
(3) $[\langle ([d, \bar{d}], \langle a; b\rangle); c\rangle, \langle \bar{a}; (\langle \bar{b}; \bar{c}\rangle, [e, \bar{e}])\rangle]$
(4) $[\langle a; (b, [d, c])\rangle, \langle \bar{a}; [\bar{b}, \langle \bar{d}; \bar{c}\rangle]\rangle]$
(5) $[\langle (b, c); [d, e]\rangle, \langle [\bar{b}, \bar{c}]; (\bar{d}, \bar{e})\rangle]$
(6) $[d, \langle (\bar{d}, \langle a; b\rangle); c\rangle, \langle \bar{a}; (\langle \bar{b}; \bar{c}\rangle, [e, \bar{e}])\rangle]$
(7) $[\langle a; [b, c]\rangle, d, (\bar{d}, \langle [\bar{a}, \bar{b}]; \bar{c}\rangle)]$
(8) $[\bar{a}, (a, \langle d; \bar{b}\rangle), (b, c), \langle \bar{d}; \bar{c}\rangle]$
(9) $[\bar{a}, \langle a; d; \bar{b}\rangle, \langle b; c\rangle, \langle \bar{d}; \bar{c}\rangle]$
(10) $[a, \langle b; d\rangle, \langle \bar{b}; c\rangle, (\bar{a}, \bar{c}, \bar{d})]$

In the results of the experiments, it is important to observe that, besides the increase in the speed of the search, number of rewrites performed differs drastically between the runs of the same search query in modules for systems BV, BVn and BVu. Moving from system BV to system BVn gets rid of the trivial instances of the inference rules. The increase in the performance while moving from BVn to BVu is due to the merging of the instances of the atomic interaction and unit rules, and also due to the absence of the instances of the switch 2 rule. The following proposition helps to understand this better.

		finds a proof		search terminates	
Query	System	in # millisec.	after # rewrites	in # millisec.	after # rewrites
1.	BV	2880	579948	3150	631564
	BVn	260	30101	260	30125
	BVu	80	7034	90	7929
2.	BV	9180	1627703	16480	2919393
	BVn	1250	134404	1280	135955
	BVu	150	14684	670	57617
3.	BV	22410	3957563	55280	9133453
	BVn	3280	295044	3290	295660
	BVu	330	27252	780	57956
4.	BV	47050	7746694	91990	14284074
	BVn	6540	605455	6570	607034
	BVu	390	36549	1370	117427
5.	BV	25820	4790935	36100	6709682
	BVn	2790	291791	2800	292229
	BVu	460	43304	1060	86880
6.	BV	227590	31623834	505330	58853999
	BVn	24590	1975556	24600	1976657
	BVu	3270	258313	6140	447774
7.	BV	421620	48377830	559550	61766550
	BVn	37020	2874128	37030	2874710
	BVu	4420	382911	7280	605438
8.	BV	525170	55377537	1054080	96078554
	BVn	60890	3695443	61040	3703525
	BVu	6200	469793	19060	1248859
9.	BV	–	–	–	–
	BVn	151870	9183688	151950	9187017
	BVu	9520	855145	18370	1568647
10.	BV	–	–	–	–
	BVn	308150	9418863	308660	9429282
	BVu	17660	1187905	70680	3129460

TABLE 4.1. Representative performance comparison of proof search in the implementations of the systems BV, BVn and BVu

PROPOSITION 4.34. *Let $R \neq \circ$ be a* BV *structure in normal form with n number of positive atoms. If R has a proof in* BVn *with length k, then R has a proof in* BVu *with length $k - n$.*

PROOF. By induction on the number of positive atoms in R: The base case is proved because $R \neq \circ$. Returning to the inductive case observe that while going up in the proof of R in BVn, each positive atom must be annihilated with its negation by an application of the rule ai↓ and then the resulting structure must be transformed

to a normal form by equivalently removing the unit $\circ$ with an application of one of the rules $\mathsf{u}_1{\downarrow}, \mathsf{u}_2{\downarrow}, \mathsf{u}_3{\downarrow}$ and $\mathsf{u}_4{\downarrow}$. In BVn these two steps are replaced by a single application of one of the rules $\mathsf{ai}_1{\downarrow}, \mathsf{ai}_2{\downarrow}, \mathsf{ai}_3{\downarrow}$ and $\mathsf{ai}_4{\downarrow}$. □

4.3.2. Equations for Unit in System NEL. In this subsection, I will carry the previous ideas on system BV to system NEL in order to remove the equations for the unit from the equational system underlying system NEL.

Definition 4.35. *The term rewriting system* $\mathsf{R}^{\mathsf{NEL}}_{Unit}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{NEL}}_{Unit} = \mathsf{R}^{\mathsf{NEL}}_{Exp} \quad \cup \quad \left\{ \begin{array}{llll} [\,R\,,\circ\,] & \rightarrow & R & \qquad [\,\circ\,,R\,] \;\rightarrow\; R \\ (\,R\,,\circ\,) & \rightarrow & R & \qquad (\,\circ\,,R\,) \;\rightarrow\; R \\ \langle\, R\,;\circ\,\rangle & \rightarrow & R & \qquad \langle\,\circ\,;R\,\rangle \;\rightarrow\; R \end{array} \right.$$

Proposition 4.36. *The term rewriting system* $\mathsf{R}^{\mathsf{NEL}}_{Unit}$ *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{NEL}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{NEL}}_{Unit}$ *is in unit normal form.*

Proof. Analogous to the proof of Proposition 4.23. □

The following functional Maude module implements the term rewriting system $\mathsf{R}^{\mathsf{NEL}}_{Unit}$.

```
fmod NEL-UNF is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op o     : -> Unit .
  op -_    : Structure -> Structure .
  op ?_    : Structure -> Structure .
  op !_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure .
  op {_,_} : Structure Structure -> Structure .
  op <_;_> : Structure Structure -> Structure .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .
     eq  - o          =  o .
     eq  - [ R , T ]  =  { - R , - T } .
     eq  - { R , T }  =  [ - R , - T ] .
     eq  - < R ; T >  =  < - R ; - T > .
     eq  - - R        =  R .

     eq  - ? R  =  ! - R .
     eq  - ! R  =  ? - R .
     eq  ? ? R  =  ? R .
     eq  ! ! R  =  ! R .

     eq  ? o  =  o .
```

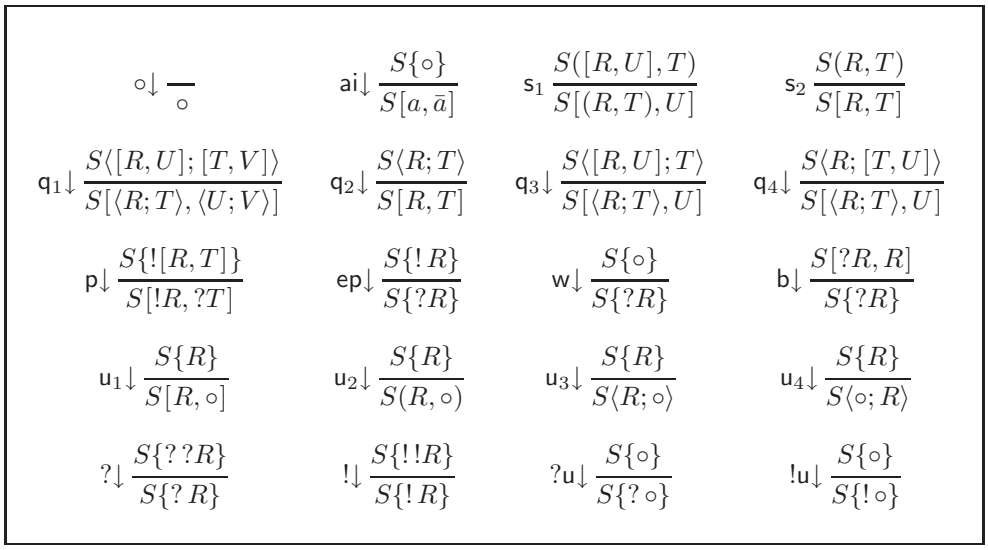


FIGURE 4.5. System NELn

```
    eq   ! o  =  o .

    eq   [ R , o ]  =  R .    eq   [ o , R ]  =  R .
    eq   { R , o }  =  R .    eq   { o , R }  =  R .
    eq   < R ; o >  =  R .    eq   < o ; R >  =  R .
endfm
```

DEFINITION 4.37. *Let* NELu *be the equational system obtained by removing the equations for the unit from the equational system* NELe.

DEFINITION 4.38. *The system shown in Figure 4.5 is called system* NELn. *The rules of this system are called* unit *($\circ\downarrow$),* atomic interaction *($\mathsf{ai}\downarrow$),* switch 1 *(s_1),* switch 2 *(s_2),* seq 1 *($\mathsf{q}_1\downarrow$),* seq 2 *($\mathsf{q}_2\downarrow$),* seq 3 *($\mathsf{q}_3\downarrow$),* seq 4 *($\mathsf{q}_4\downarrow$),* promotion *($\mathsf{p}\downarrow$),* exponential promotion *($\mathsf{ep}\downarrow$),* weakening *($\mathsf{w}\downarrow$),* absorption *($\mathsf{b}\downarrow$),* unit 1 *($\mathsf{u}_1\downarrow$),* unit 2 *($\mathsf{u}_2\downarrow$),* unit 3 *($\mathsf{u}_3\downarrow$),* unit 4 *($\mathsf{u}_4\downarrow$),* why not *($?\downarrow$),* of course *($!\downarrow$),* why not unit *($?\mathsf{u}\downarrow$), and* of course unit *($!\mathsf{u}\downarrow$). Inference rules of system* NELn *are applied on* NEL *structures, which are considered equivalent modulo the equational system* NELu.

PROPOSITION 4.39. *Every* NEL *structure in exponential normal form can be transformed to a structure in unit normal form by applying the rules* $\{\mathsf{u}_1\downarrow, \mathsf{u}_2\downarrow, \mathsf{u}_3\downarrow, \mathsf{u}_4\downarrow\}$ *in Figure 4.5 bottom-up.*

PROOF. Similar to the proof of Proposition 4.26, the result follows immediately from Proposition 4.36. □

THEOREM 4.40. *System* NEL *and system* NELn *are strongly equivalent.*

PROOF. It is immediate that all the inference rules of system NELn are derivable for system NEL. For the proof of the other direction, from Proposition 4.8, we have that every derivation in NEL can be rewritten as a derivation Δ in NELe. Let

Δ be the derivation $\begin{array}{c} W \\ \Delta \| \mathsf{NELe} \\ Q \end{array}$. From Proposition 4.39, we get a unit normal form of Q while going up in a derivation. With induction on the length of Δ, we will construct a derivation

$$\begin{array}{c} W' \\ \Delta' \| \mathsf{NELn} \\ Q' \end{array}$$

where W' is a unit normal form of W: The base case and the cases for the inference rules $\mathsf{ai}{\downarrow}$, s and $\mathsf{q}{\downarrow}$ are same as in Lemma 4.27.

- If $\mathsf{p}{\downarrow}\dfrac{P}{Q}$ is the last rule applied in Δ where $Q = S[!R, ?T]$ for a context S and structures R and T, then by induction hypothesis there is a derivation $\begin{array}{c} W \\ \| \mathsf{NELn} \\ P \end{array}$. The following cases exhaust the possibilities:
  - If $R \neq \circ$ and $T \neq \circ$, then take the same instance of $\mathsf{p}{\downarrow}$ as in Δ.
  - If $R = \circ$ and $T \neq \circ$, then $Q = S'[?T]$ where S' is a normal form of context S. Then $P = S'\{!T\}$. Apply the rule $\mathsf{ep}{\downarrow}$ to Q.
  - Other 2 cases, where $R \neq \circ$ and $T = \circ$, or $R = \circ$ and $T = \circ$, are trivial instances of the $\mathsf{p}{\downarrow}$ rule. Take $P = Q$.
- If $\rho\dfrac{P}{Q}$ is the last rule applied in Δ where $\rho \in \{\mathsf{w}{\downarrow}, \mathsf{b}{\downarrow}, ?{\downarrow}, !{\downarrow}\}$ such that $Q = S'\{?R\}$ or $Q = S'\{!R\}$, and $R = \circ$, then these instances are trivial instances of the rule ρ. Take $P = Q$. Otherwise, apply the rule ρ as in Δ.

□

The Maude system module below implements system NELn.

```
mod NELn is
 sorts Atom Unit Structure .
 subsort Atom < Structure .
 subsort Unit < Structure .

  op o      : -> Unit .
  op -_     : Atom -> Atom [ prec 50 ] .
  op ?_     : Structure -> Structure [ prec 60 ] .
  op !_     : Structure -> Structure [ prec 60 ] .
  op [_,_]  : Structure Structure -> Structure [assoc comm] .
  op {_,_}  : Structure Structure -> Structure [assoc comm] .
  op <_;_>  : Structure Structure -> Structure [assoc] .

  ops a b c d e f g h i j : -> Atom .

 var R T U V : Structure .
 var A : Atom .
```

```
rl [ai] : [ A , - A ]  => o .

rl [switch-1] :  [ { R , T } , U ]    => { [ R , U ] , T } .
rl [switch-2] :  [ R , T ]            => { R, T } .

rl [seq-1] : [ < R ; T > , < U ; V > ] => < [R,U] ; [T,V] > .
rl [seq-2] : [ R , T ]                  => < R ; T > .
rl [seq-3] : [ < R ; T > , U ]          => < [ R , U ] ; T > .
rl [seq-4] : [ < R ; T > , U ]          => < R ; [ T , U ] > .

rl [promotion]    :  [ ! R , ? T ]  =>  ! [ R , T ] .
rl [e-promotion]  :  ? R            =>  ! R .
rl [weakening]    :  ? R            =>  o .
rl [absorption]   :  ? R            =>  [ ? R , R ] .

rl [unit-1] :  [ R , o ]   =>  R .
rl [unit-2] :  { R , o }   =>  R .
rl [unit-3] :  < R ; o >   =>  R .
rl [unit-4] :  < o ; R >   =>  R .

rl [why-not]    :  ? R  => ? ? R .
rl [of-course]  :  ! R  => ! ! R .

rl [wn-unit]  :  ? o  =>  o .
rl [oc-unit]  :  ! o  =>  o .
endm
```

Analogous to system BV, the equations for the unit in system NEL can be also equivalently removed from the language of this logic. In order to do this, the inference rules with the unit in the premise should be placed in all the possible contexts, including the modalities. It is well known that linear logic has 7 modalities, that is, empty modality, !, ?, !?, ?!, ?!? and !?!. Thus, by introducing new inference rules also for each of these modalities, an equivalent system to system NELn can be designed.

4.3.3. Equations for Units in System LS. In this subsection I will present a system, called system LSn, which is equivalent to system LS. The equations for units are redundant for system LSn while proving provable LS structures, thus they will be removed from the underlying equational system:

Definition 4.41. *Let* ELSu *be the equational system obtained by removing the equations for units from the equational system* ELSe.

Definition 4.42. *The term rewriting system* $\mathsf{R}^{\mathsf{LS}}_{Unit}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{LS}}_{Unit} = \mathsf{R}^{\mathsf{LS}}_{Exp} \cup \left\{ \begin{array}{lcllcl} [\bot, R] & \to & R & \quad \{0, R\} & \to & R \\ (1, R) & \to & R & \quad \langle \top, R \rangle & \to & R \\ \{\bot, \bot\} & \to & \bot & \quad \langle 1, 1 \rangle & \to & 1 \end{array} \right.$$

PROPOSITION 4.43. *The term rewriting system is* $\mathsf{R}^{\mathsf{LS}}_{Unit}$ *modulo* ELSu *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{LS}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{LS}}_{Unit}/\mathsf{ELSu}$ *is in unit normal form.*

PROOF. Observe that the system ELSu consists of equations for associativity and commutativity for the binary function symbols. Thus a Σ_{LS}-term can be considered in canonical form which is defined by a total lexical order

$$\bot < \mathsf{1} < \mathsf{0} < \top < a \qquad \text{(where } a \text{ denotes any atom)}$$

such that the units $\bot$, $\mathsf{1}$, $\mathsf{0}$, and $\top$ appear always on the left of the atoms and the same units appear next to each other as the arguments of the same binary function symbol in a Σ_{LS}-term. The rest of the proof is same as Proposition 3.24. □

The following functional Maude module implements the term rewriting system $\mathsf{R}^{\mathsf{LS}}_{Unit}/\mathsf{ELSu}$.

```
fmod LS-UNF is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op 1     : -> Unit .
  op bot   : -> Unit .
  op 0     : -> Unit .
  op top   : -> Unit .

  op -_      : Structure -> Structure .
  op ?_      : Structure -> Structure .
  op !_      : Structure -> Structure .
  op [_,_]   : Structure Structure -> Structure [assoc comm] .
  op {_,_}   : Structure Structure -> Structure [assoc comm] .
  op [|_,_|] : Structure Structure -> Structure [assoc comm] .
  op {|_,_|} : Structure Structure -> Structure [assoc comm] .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

     eq  - bot   =  1 .
     eq  - 1     =  bot .
     eq  - top   =  0 .
     eq  - 0     =  top .

     eq  - [ R , T ]    =  { - R , - T } .
     eq  - { R , T }    =  [ - R , - T ] .
     eq  - [| R , T |]  =  {| - R , - T |} .
     eq  - {| R , T |}  =  [| - R , - T |] .
     eq  - - R          =  R .

     eq  [ R , bot ]      =  R .
     eq  { R , 1 }        =  R .
```

```
    eq  [| R , 0 |]      =  R.
    eq  {| R , top |}    =  R.
    eq  [| bot , bot |]  =  bot .
    eq  {| 1 , 1 |}      =  1 .

    eq  - ? R  =  ! - R .
    eq  - ! R  =  ? - R .

    eq  ? ? R  =  ? R .
    eq  ! ! R  =  ! R .

    eq  ? bot  = bot .
    eq  ! 1    = bot .
endfm
```

DEFINITION 4.44. *The system shown in Figure 4.6 is called system* LSn*. In addition to the inference rules that are common with system* LSe*, the rules of this system are called* unit 1 *(*$\mathsf{u}_1{\downarrow}$*),* unit 2 *(*$\mathsf{u}_2{\downarrow}$*),* unit 3 *(*$\mathsf{u}_3{\downarrow}$*),* unit 4 *(*$\mathsf{u}_4{\downarrow}$*),* unit 5 *(*$\mathsf{u}_3{\downarrow}$*), and* unit 6 *(*$\mathsf{u}_4{\downarrow}$*). The inference rules of system* LSn *are applied on* LS *structures which are considered equivalent modulo the equational system* ELSu*.*

$$1{\downarrow}\,\frac{}{1} \qquad \mathsf{ai}{\downarrow}\,\frac{S\{1\}}{S[a,\bar{a}]} \qquad \mathsf{s}\,\frac{S([R,U],T)}{S[(R,T),U]}$$

$$\mathsf{t}{\downarrow}\,\frac{S\{0\}}{S\{R\}} \qquad \mathsf{c}{\downarrow}\,\frac{S\{R,R\}}{S\{R\}} \qquad \mathsf{d}{\downarrow}\,\frac{S\langle[R,U],[T,V]\rangle}{S[\langle R,T\rangle,\{U,V\}]}$$

$$\mathsf{w}{\downarrow}\,\frac{S\{\bot\}}{S\{?R\}} \qquad \mathsf{b}{\downarrow}\,\frac{S[?R,R]}{S\{?R\}} \qquad \mathsf{p}{\downarrow}\,\frac{S\{![R,T]\}}{S[!R,?T]}$$

$$\mathsf{u}_1{\downarrow}\,\frac{S\{R\}}{S(1,R)} \qquad \mathsf{u}_2{\downarrow}\,\frac{S\{R\}}{S[\bot,R]} \qquad \mathsf{u}_3{\downarrow}\,\frac{S\{R\}}{S\{0,R\}}$$

$$\mathsf{u}_4{\downarrow}\,\frac{S\{R\}}{S\langle\top,R\rangle} \qquad \mathsf{u}_5{\downarrow}\,\frac{S\{1\}}{S\langle 1,1\rangle} \qquad \mathsf{u}_6{\downarrow}\,\frac{S\{\bot\}}{S\{\bot,\bot\}}$$

$$?\mathsf{u}{\downarrow}\,\frac{S\{\bot\}}{S\{?\bot\}} \qquad !\mathsf{u}{\downarrow}\,\frac{S\{1\}}{S\{!\,1\}} \qquad ?{\downarrow}\,\frac{S\{??R\}}{S\{?R\}}$$

FIGURE 4.6. System LSn

I will now show that system LSn is complete for linear logic. For this purpose, I am going to employ the sequent calculus presentation of linear logic.

PROPOSITION 4.45. *Every* LS *structure in exponential normal form can be transformed to a structure in unit normal form by applying the rules* $\{\mathsf{u}_1\downarrow, \mathsf{u}_2\downarrow, \mathsf{u}_3\downarrow, \mathsf{u}_4\downarrow, \mathsf{u}_5\downarrow, \mathsf{u}_6\downarrow\}$ *in Figure 4.6 bottom-up.*

PROOF. Follows immediately from Proposition 4.43. □

The following proposition and the theorem thereafter are restricted versions of similar results in [**Str03a**] which I carry from system LS to LSn.

PROPOSITION 4.46. *Let* R *be an* LS *structure in unit normal form. The rule* $\mathsf{i}\downarrow \dfrac{S\{1\}}{S[R, \bar{R}]}$ *is derivable in the system* $\{\mathsf{ai}\downarrow, \mathsf{s}, \mathsf{d}\downarrow, \mathsf{p}\downarrow, \mathsf{u}_1\downarrow, \mathsf{u}_5\downarrow, !\mathsf{u}\downarrow\}$ *where the underlying equational system is* ELSu.

PROOF. For a given application of $\mathsf{i}\downarrow$ by structural induction on R, we will construct an equivalent derivation that contains only the instances of the above rules.

- R is an atom: Then the given instance of $\mathsf{i}\downarrow$ is an instance of $\mathsf{ai}\downarrow$.
- if $R = [P, Q]$, then apply the induction hypothesis to

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{S\{1\}}{S[Q,\bar{Q}]}\,\mathsf{i}\downarrow}{S(1,[Q,\bar{Q}])}\,\mathsf{u}_1\downarrow}{S([P,\bar{P}],[Q,\bar{Q}])}\,\mathsf{i}\downarrow}{S[P,(\bar{P},[Q,\bar{Q}])]}\,\mathsf{s}}{S[P,Q,(\bar{P},\bar{Q})]}\,\mathsf{s} \quad .$$

- if $R = (P, Q)$, then it is similar to the previous case.
- if $R = \{P, Q\}$, then apply the induction hypothesis to

$$\dfrac{\dfrac{\dfrac{\dfrac{S\{1\}}{S\langle 1,1\rangle}\,\mathsf{u}_5\downarrow}{S\langle 1,[Q,\bar{Q}]\rangle}\,\mathsf{i}\downarrow}{S\langle [P,\bar{P}],[Q,\bar{Q}]\rangle}\,\mathsf{i}\downarrow}{S[\{P,Q\},\langle\bar{P},\bar{Q}\rangle]}\,\mathsf{d}\downarrow \quad .$$

- if $R = \langle P, Q\rangle$, then it is similar to the previous case.
- if $R = ?\,P$, then apply the induction hypothesis to

$$\dfrac{\dfrac{\dfrac{S\{1\}}{S\{!\,1\}}\,!\mathsf{u}\downarrow}{S\{!\,[P,\bar{P}]\}}\,\mathsf{i}\downarrow}{S[?\,P,!\,\bar{P}]}\,\mathsf{p}\downarrow \quad .$$

- if $R = !\,P$, then it is similar to the previous case.

□

THEOREM 4.47. *Let* $\vdash \Phi$ *be a sequent and* $\underline{\vdash \Phi}_{\mathsf{s}}$ *be the* LS *structure, obtained from* $\vdash \Phi$ *by the translation function* $\underline{\cdot}_{\mathsf{s}}$. *If* $\vdash \Phi$ *is cut-free provable in* LL, *then the structure* $\underline{\vdash \Phi}_{\mathsf{s}}$ *is provable in* LSn.

PROOF. Let Π be a proof of $\vdash \Phi$ in LL. By structural induction on Π, we will construct a proof $\underline{\Pi}_{\mathsf{s}}$ of $\vdash \underline{\Phi}_{\mathsf{s}}$ in system LSn.

- If Π is $\mathsf{id}\dfrac{}{\vdash A, A^{\perp}}$ for some formula A, then let $\underline{\Pi}_{\mathsf{s}}$ be the proof obtained via Proposition 4.46 from

$$\mathsf{i}{\downarrow}\dfrac{1{\downarrow}\dfrac{}{1}}{[\underline{A}_{\mathsf{s}}, \overline{\underline{A}_{\mathsf{s}}}]} \quad .$$

- If $\parr\dfrac{\vdash A, B, \Phi}{\vdash A \parr B, \Phi}$ is the last rule applied in Π, then let $\underline{\Pi}_{\mathsf{s}}$ be the proof of $[\underline{A}_{\mathsf{s}}, \underline{B}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}]$ that exists by induction hypothesis.

- If $\otimes\dfrac{\vdash A, \Phi \quad \vdash B, \Psi}{\vdash A \otimes B, \Phi, \Psi}$ is the last rule applied in Π, then there are by induction hypothesis two derivations $\begin{array}{c}1\\ \Delta_1 \,\vdots\, \mathsf{LSn}\\ [\underline{A}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}]\end{array}$ and $\begin{array}{c}1\\ \Delta_2 \,\vdots\, \mathsf{LSn}\\ [\underline{B}_{\mathsf{s}}, \underline{\Psi}_{\mathsf{s}}]\end{array}$. Let $\underline{\Pi}_{\mathsf{s}}$ be the proof

$$\begin{array}{c}
1{\downarrow}\dfrac{}{1}\\
\Delta_2 \,\Vdots\, \mathsf{LSn}\\
\mathsf{u}_1{\downarrow}\dfrac{[\underline{A}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}]}{([\underline{A}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}], 1)}\\
\Delta_1 \,\Vdots\, \mathsf{LSn}\\
\mathsf{s}\dfrac{([\underline{A}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}], [\underline{B}_{\mathsf{s}}, \underline{\Psi}_{\mathsf{s}}])}{[([\underline{A}_{\mathsf{s}}, \underline{\Phi}_{\mathsf{s}}], \underline{B}_{\mathsf{s}}), \underline{\Psi}_{\mathsf{s}}]}\\
\mathsf{s}\dfrac{}{[(\underline{A}_{\mathsf{s}}, \underline{B}_{\mathsf{s}}), \underline{\Phi}_{\mathsf{s}}, \underline{\Psi}_{\mathsf{s}}]}
\end{array} \quad .$$

- If $\bot\dfrac{\vdash \Phi}{\vdash \bot, \Phi}$ is the last rule applied in Π. Then $\underline{\Pi}_{\mathsf{s}}$ is the proof

$$\begin{array}{c}
\Pi' \,\Vdots\, \mathsf{LSn}\\
\mathsf{u}_2{\downarrow}\dfrac{\underline{\Phi}_{\mathsf{s}}}{[\bot, \underline{\Phi}_{\mathsf{s}}]}
\end{array}$$

  where Π' exists by induction hypothesis.

- If Π is $1\dfrac{}{\vdash 1}$, then let $\underline{\Pi}_{\mathsf{s}}$ be $1{\downarrow}\dfrac{}{1}$.

- If $\&\,\dfrac{\vdash A, \Phi \quad \vdash B, \Phi}{\vdash A \,\&\, B, \Phi}$ is the last rule applied in Π, then there are by induction hypothesis two derivations $\overset{1}{\underset{[\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}{\Delta_1 \,\|\!|\, \mathsf{LSn}}}$ and $\overset{1}{\underset{[\underline{B}_{\mathsf{S}}, \underline{\Psi}_{\mathsf{S}}]}{\Delta_2 \,\|\!|\, \mathsf{LSn}}}$. Let $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1}\,1\downarrow}{\langle 1, 1\rangle}\,\mathsf{u}_5\downarrow}{\langle [\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}], 1\rangle}\,\Delta_2 \,\|\!|\, \mathsf{LSn}}{\langle [\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}], [\underline{B}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]\rangle}\,\Delta_1 \,\|\!|\, \mathsf{LSn}}{[\langle \underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}\rangle, [\underline{\Phi}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]]}\,\mathsf{d}\downarrow}{[\langle \underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}\rangle, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{c}\downarrow} \quad .$$

- If $\oplus_1\,\dfrac{\vdash A, \Phi}{\vdash A \oplus B, \Phi}$ is the last rule applied in Π, then $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{\Pi' \,\|\!|\, \mathsf{LSn}}{[\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{u}_3\downarrow}{[[\underline{A}_{\mathsf{S}}, 0], \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{t}\downarrow}{[[\underline{A}_{\mathsf{S}}, \underline{B}_{\mathsf{S}}], \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{t}\downarrow$$

  where Π' exists by induction hypothesis.

- The case for the rule $\oplus_2\,\dfrac{\vdash B, \Phi}{\vdash A \oplus B, \Phi}$ is similar.

- If Π is $\top\,\dfrac{}{\vdash \top, \Phi}$, then let $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{}{1}\,1\downarrow}{[\top, 0]}\,\mathsf{ai}\downarrow}{[\top, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{t}\downarrow \quad .$$

- If $?\mathsf{d}\dfrac{\vdash A, \Phi}{\vdash ?A, \Phi}$ is the last rule applied in Π, then let $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi'\\ \|\\ \mathsf{LSn}\end{array}}{[\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{w}\downarrow}{[?\underline{A}_{\mathsf{S}}, \underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{b}\downarrow}{[?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}$$

  where Π' exists by induction hypothesis.

- If $?\mathsf{c}\dfrac{\vdash ?A, ?A, \Phi}{\vdash ?A, \Phi}$ is the last rule applied in Π, then let $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi'\\ \|\\ \mathsf{LSn}\end{array}}{[?\underline{A}_{\mathsf{S}}, ?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{w}\downarrow}{[??\underline{A}_{\mathsf{S}}, ?\underline{A}_{\mathsf{S}}, ?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{b}\downarrow}{[??\underline{A}_{\mathsf{S}}, ?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{b}\downarrow}{[??\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}\,?\downarrow}{[?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}$$

  where Π' exists by induction hypothesis.

- If $?\mathsf{w}\dfrac{\vdash \Phi}{\vdash ?A, \Phi}$ is the last rule applied in Π, then let $\underline{\Pi}_{\mathsf{S}}$ be the proof

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\Pi'\\ \|\\ \mathsf{LSn}\end{array}}{\underline{\Phi}_{\mathsf{S}}}\,\mathsf{u}_2\downarrow}{[\bot, \underline{\Phi}_{\mathsf{S}}]}\,\mathsf{w}\downarrow}{[?\underline{A}_{\mathsf{S}}, \underline{\Phi}_{\mathsf{S}}]}$$

  where Π' exists by induction hypothesis.

- If $!\dfrac{\vdash\ A, ?B_1, \ldots, ?B_n}{\vdash !A, ?B_1, \ldots, ?B_n}$ is the last rule applied in Π, then there exists by induction hypothesis a derivation $\begin{array}{c}1\\ \Delta_1\,\|\mathsf{LSn}\\ [\underline{A}_{\mathsf{S}}, ?\underline{B_1}_{\mathsf{S}}, \ldots, ?\underline{B_n}_{\mathsf{S}}]\end{array}$. Now let $\underline{\Pi}_{\mathsf{S}}$ be

the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1}\,1\downarrow}{!1}\,!\mathsf{u}\downarrow}{\Delta_1 \Big\| \mathsf{LSn}}}{![\underline{A}_{\mathsf{S}}, ?\underline{B_1}_{\mathsf{S}}, ?\underline{B_2}_{\mathsf{S}}, \ldots, ?\underline{B_n}_{\mathsf{S}}]}\,\mathsf{p}\downarrow}{[![\underline{A}_{\mathsf{S}}, ?\underline{B_1}_{\mathsf{S}}, \ldots, ?\underline{B_{n-1}}_{\mathsf{S}}], ??\underline{B_n}_{\mathsf{S}}]}\,?\downarrow}{\vdots}\,?\downarrow}{[![\underline{A}_{\mathsf{S}}, ?\underline{B_1}_{\mathsf{S}}], ?\underline{B_2}_{\mathsf{S}}, \ldots, ?\underline{B_n}_{\mathsf{S}}]}\,\mathsf{p}\downarrow}{[!\underline{A}_{\mathsf{S}}, ??\underline{B_1}_{\mathsf{S}}, ?\underline{B_2}_{\mathsf{S}}, \ldots, ?\underline{B_n}_{\mathsf{S}}]}\,?\downarrow}{[!\underline{A}_{\mathsf{S}}, ?\underline{B_1}_{\mathsf{S}}, ?\underline{B_2}_{\mathsf{S}}, \ldots, ?\underline{B_n}_{\mathsf{S}}]}\ .$$

□

COROLLARY 4.48. *Systems* LS *and* LSn *are equivalent.*

PROOF. It is immediate that the rules of system LSn are derivable for system LS. Result follows from Proposition 4.45 and Theorem 4.47. □

The Maude system module below implements system LSn.

```
mod LSn is
  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  op 1       : -> Unit .
  op bot     : -> Unit .
  op 0       : -> Unit .
  op top     : -> Unit .

  op -_      : Atom -> Atom [ prec 50 ] .
  op ?_      : Structure -> Structure [ prec 60 ] .
  op !_      : Structure -> Structure [ prec 60 ] .
  op [_,_]   : Structure Structure -> Structure [assoc comm] .
  op {_,_}   : Structure Structure -> Structure [assoc comm] .
  op [|_,_|] : Structure Structure -> Structure [assoc comm] .
  op {|_,_|} : Structure Structure -> Structure [assoc comm] .

  ops a b c d e f g h l : -> Atom .

  var R T U V : Structure .
  var A : Atom .

    rl [ai-down]  : [ A , - A ]         => 1 .
    rl [switch]   : [ { R , T } , U ]   => { [ R , U ] , T } .
```

```
    rl [additive] : [ {| R , T |} , [| U , V |] ]
                            =>   {| [ R , U ] , [ T , V ]  |} .
    rl [thinning]    : R    =>   0 .
    rl [contraction] : R    =>   [| R , R |] .

    rl [promotion] : [ ! R , ? T ]  =>  ! [ R , T ] .
    rl [weakening] : ? R            =>  bot .
    rl [absorption]: ? R            =>  [ ? R , R ] .

    rl [unit-1]    : { 1 , R }    =>  R .
    rl [unit-2]    : [ bot , R ]  =>  R .
    rl [unit-3]    : [| 0 , R |]  =>  R .

    rl [unit-4]    : {| top , R |}    =>  R .
    rl [unit-5]    : {| 1 , 1 |}      =>  1 .
    rl [unit-6]    : [| bot , bot |]  =>  bot .

    rl [?-unit]    : ? bot       =>  bot .
    rl [!-unit]    : ! 1         =>  1 .
    rl [why-not]   : ? R         =>  ? ? R .
    rl [ai-unit]   : [ top , 0 ] => 1 .
endm
```

4.3.4. Equations for Units in System KSg. In this subsection, I will present a system equivalent to system KSg, for which the equations for units become redundant. I will then present a system for classical logic in the calculus of structures, where the underlying equational system can be completely removed.

DEFINITION 4.49. *Let* EKSgu *be the equational system obtained by removing the equations for units from the equational system* EKSg.

DEFINITION 4.50. *The term rewriting system* $\mathsf{R}^{\mathsf{KSg}}_{Unit}$ *is defined as follows:*

$$\mathsf{R}^{\mathsf{KSg}}_{Unit} = \mathsf{R}^{\mathsf{KSg}}_{Neg} \quad \cup \quad \left\{ \begin{array}{lcllcl} [\mathsf{ff}, R] & \to & R & [\mathsf{tt}, \mathsf{tt}] & \to & \mathsf{tt} \\ (\mathsf{tt}, R) & \to & R & (\mathsf{ff}, \mathsf{ff}) & \to & \mathsf{ff} \end{array} \right.$$

PROPOSITION 4.51. *The term rewriting system is* $\mathsf{R}^{\mathsf{KSg}}_{Unit}$ *modulo* EKSg *is (i) terminating and (ii) confluent. (iii) Let* s *be a* Σ_{KSg}*-term. The normal form of* s *with respect to* $\mathsf{R}^{\mathsf{KSg}}_{Unit}/\mathsf{EKSg}$ *is in unit normal form.*

PROOF. Observe that the system EKSgu consists of equations for associativity and commutativity for the binary function symbols for conjunction and disjunction. Thus a Σ_{KSg}-term can be considered in canonical form which is defined by a total lexical order

$$\mathsf{ff} < \mathsf{tt} < a \qquad \text{(where } a \text{ denotes any atom)}$$

such that the units ff, and tt appear always on the left-hand-side of the atoms and the same units appear next to each other as the arguments of the same binary function symbol in a Σ_{KSg}-term. The rest of the proof is same as Proposition 3.24. □

```
fmod KSg-UNF is
  sorts Unit Atom Structure .

  subsort Unit < Structure .
  subsort Atom < Structure .

  op tt    : -> Unit .
  op ff    : -> Unit .
  op -_    : Structure -> Structure .
  op [_,_] : Structure Structure -> Structure [assoc comm] .
  op {_,_} : Structure Structure -> Structure [assoc comm] .

  ops a b c d e f g h i j : -> Atom .

  var R T U : Structure .

     eq  - tt         =  ff .
     eq  - ff         =  tt .
     eq  - [ R , T ]  =  { - R , - T } .
     eq  - { R , T }  =  [ - R , - T ] .
     eq  - - R        =  R .

     eq  [ ff , R ]   =  R .
     eq  { tt , R }   =  R .
     eq  [ tt , tt ]  =  tt .
     eq  { ff , ff }  =  ff .
endfm
```

DEFINITION 4.52. *The system shown in Figure 4.7 is called system* KSgn*. In addition to the inference rules that are common with system* KSg*, the rules of this system are called* unit 1 unit 1 *(*$\mathsf{u}_1{\downarrow}$*), and* unit 2 *(*$\mathsf{u}_2{\downarrow}$*). The inference rules of system* KSgn *are applied on* KSg *structures which are considered equivalent modulo the equational system* EKSgu*.*

PROPOSITION 4.53. *Systems* KSg *and* KSgn *are equivalent.*

PROOF. Observe that the rules of system KSgn are derivable for system KSg. The other direction of the proof is similar to the semantic cut elimination for system

$$\mathsf{tt}{\downarrow}\,\frac{}{\mathsf{tt}} \qquad \mathsf{ai}{\downarrow}\,\frac{S\{\mathsf{tt}\}}{S[a,\bar{a}]} \qquad \mathsf{s}\,\frac{S([R,U],T)}{S[(R,T),U]} \qquad \mathsf{w}{\downarrow}\,\frac{S\{\mathsf{ff}\}}{S\{R\}}$$

$$\mathsf{c}{\downarrow}\,\frac{S[R,R]}{S\{R\}} \qquad \mathsf{u}_1{\downarrow}\,\frac{S\{R\}}{S[R,\mathsf{ff}]} \qquad \mathsf{u}_2{\downarrow}\,\frac{S\{R\}}{S(R,\mathsf{tt})}$$

FIGURE 4.7. System KSgn

KSg in [**Brü03b**]: the invertible rule

$$\mathsf{d}\,\frac{S([R,U],[T,U])}{S[(R,T),U]}$$

is derivable for the rules s and c↓:

$$\begin{array}{c}\mathsf{s}\,\dfrac{S([R,U],[T,U])}{S[([R,U],T),U]}\\ \mathsf{c}{\downarrow}\,\dfrac{S[([R,U],T),U]}{S[(R,T),U]}\end{array}$$

Apply this rule exhaustively to obtain the conjunctive normal form of the structure. The rest of the proof is as in the proof of Theorem 4.56. □

In fact, by exploiting the contraction and weakening rules, it is possible to design a system for classical logic in the calculus of structures where the equational theory underlying structures becomes redundant. Below I will give such a system which resembles sequent calculus system G3 given in [**TS96**].

Definition 4.54. *The system shown in Figure 4.8 is called system* DKSg*. The rules of the system are called* axiom *(*t↓*),* atomic interaction *(*ai↓*),* distributivity left *(*dl*),* distributivity right *(*dr*),* unit 1 left *(*$\mathsf{u}_{1l}{\downarrow}$*),* unit 1 right *(*$\mathsf{u}_{1r}{\downarrow}$*),* unit 2 left *(*$\mathsf{u}_{2l}{\downarrow}$*), and* unit 2 right *(*$\mathsf{u}_{2r}{\downarrow}$*).*

The inference rules of the system DKSg are applied on the KSg structures. However, this system does not require an underlying equational theory for the KSg structures. In the following, I will show that system DKSg is complete for classical logic, in a way similar to the semantic cut elimination proof for system KSg in [**Brü03b**].

Lemma 4.55. *For a structure R there exists a derivation* $\begin{array}{c}R'\\ \|\{\mathsf{dl},\mathsf{dr}\}\\ R\end{array}$ *such that R' is in conjunctive normal form.*

Proof. With structural induction on R.

- If $R = a$ then R is in conjunctive normal form.

$$\mathsf{t}{\downarrow}\,\frac{}{\mathsf{t}} \qquad \mathsf{ai}{\downarrow}\,\frac{S\{\mathsf{t}\}}{S[a,\bar{a}]} \qquad \mathsf{dl}\,\frac{S([U,R],[U,T])}{S[U,(R,T)]} \qquad \mathsf{dr}\,\frac{S([R,U],[T,U])}{S[(R,T),U]}$$

$$\mathsf{w}{\downarrow}\,\frac{S\{\mathsf{f}\}}{S\{R\}} \qquad \mathsf{u}_{1l}{\downarrow}\,\frac{S\{R\}}{S[\mathsf{f},R]} \qquad \mathsf{u}_{1r}{\downarrow}\,\frac{S\{R\}}{S[R,\mathsf{f}]} \qquad \mathsf{u}_{2l}{\downarrow}\,\frac{S\{R\}}{S(\mathsf{t},R)} \qquad \mathsf{u}_{2r}{\downarrow}\,\frac{S\{R\}}{S(R,\mathsf{t})}$$

Figure 4.8. System DKSg

- If $R = (R_1, R_2)$ then with the induction hypothesis, there are derivations $\begin{array}{l} R_1' \\ \Delta_1 \,\|\{\mathsf{dl},\mathsf{dr}\} \\ R_1 \end{array}$ and $\begin{array}{l} R_2' \\ \Delta_2 \,\|\{\mathsf{dl},\mathsf{dr}\} \\ R_2 \end{array}$ where R_1' and R_2' are in conjunctive normal form. It follows that there is a derivation
$$\begin{array}{l} (R_1', R_2') \\ \Delta_2 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ (R_1', R_2) \\ \Delta_1 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ (R_1, R_2) \end{array}$$
where (R_1', R_2') is in conjunctive normal form.
- If $R = [R_1, R_2]$ then with the induction hypothesis, there are derivations $\begin{array}{l} R_1' \\ \Delta_1 \,\|\{\mathsf{dl},\mathsf{dr}\} \\ R_1 \end{array}$ and $\begin{array}{l} R_2' \\ \Delta_2 \,\|\{\mathsf{dl},\mathsf{dr}\} \\ R_2 \end{array}$ where R_1' and R_2' are in conjunctive normal form.

  Let $R_1' = (S_1, S_2)$, and $R_2' = (T_1, T_2)$ where S_1, S_2, T_1, T_2 are in conjunctive normal form. Consider the following derivation.
$$\mathsf{dr}\,\dfrac{\mathsf{dr}\,\dfrac{\mathsf{dl}\,\dfrac{(([S_1,T_1],[S_2,T_1]),([S_1,T_2],[S_2,T_2]))}{(([S_1,T_1],[S_2,T_1]),[(S_1,S_2),T_2])}}{([(S_1,S_2),T_1],[(S_1,S_2),T_2])}}{[(S_1,S_2),(T_1,T_2)]}\ .$$
  By applying the rules $\mathsf{dl}; \mathsf{dr}; \mathsf{dr}$ to each conjunct $[S_i, T_j]$ as many times as possible, we get the derivations
$$\begin{array}{l} R_3' \\ \Delta_3 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ [S_1, T_1] \end{array}\ ,\quad \begin{array}{l} R_4' \\ \Delta_3 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ [S_2, T_1] \end{array}\ ,\quad \begin{array}{l} R_5' \\ \Delta_3 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ [S_1, T_2] \end{array}\ ,\quad \begin{array}{l} R_6' \\ \Delta_3 \,\Big\|\{\mathsf{dl},\mathsf{dr}\} \\ [S_2, T_2] \end{array}$$
  where R_3', R_4', R_5', R_6' are in conjunctive normal form.

□

THEOREM 4.56. *A structure R has a proof in* KSg *if and only if there are structures T' and T'' and a proof in* DKSg *such that*
$$\begin{array}{l} \top \\ \Big\|\{\mathsf{ai}{\downarrow},\mathsf{u}_{2l}{\downarrow},\mathsf{u}_{2r}{\downarrow}\} \\ T'' \\ [\,|\{\mathsf{w}{\downarrow},\mathsf{u}_{1l}{\downarrow},\mathsf{u}_{1r}{\downarrow}\} \\ T' \\ [\,|\{\mathsf{dl},\mathsf{dr}\} \\ R \end{array}\ .$$

PROOF. Observe that the rules of system DKSg, including dl and dr are derivable for system KSg, as it can be seen in the proof of Proposition 4.53. For the proof of the other direction, from Lemma 4.55 it follows that there is a structure

T' which is in conjunctive normal form. The rules dl and dr are invertible rules, so T' is also provable in KSg. Because T' is in conjunctive normal form, constructed by binary connectives, T' must be of the form

$$T' = Q\{R_k\}$$

where R_k is a disjunction of atoms. For T' to be provable in KSg, each such R_i must be of one of the following forms

$$[P_1\{a\}, P_2\{\bar{a}\}]$$

where $P_1\{\ \}$ and $P_2\{\ \}$ are nested disjunctions with a hole. Obviously for every such disjunction, there is a derivation

$$\begin{array}{c} [a, \bar{a}] \\ \Delta_k \Big\| \{\mathsf{w}\downarrow, \mathsf{u}_{1l}\downarrow, \mathsf{u}_{1r}\downarrow\} \\ [P_1\{a\}, P_2\{\bar{a}\}] \end{array}$$

which leads to T''. Applying the rule $\mathsf{ai}\downarrow$, and then $\mathsf{u}_{2l}\downarrow$ and $\mathsf{u}_{2r}\downarrow$ to each disjunction in T'' we get a proof. □

Similar to the systems which are discussed in the previous sections, system DKSg can be implemented in Maude. However, the decomposition of the proofs in this system, given in Theorem 4.56, can be used as a proof strategy while proving structures. This can be achieved in system DKSg because of the availability of normal forms (for instance conjunctive normal form) at the end of each phase of the proof. I employ the meta-level features of the language Maude, which allows to represent such a decomposition as meta-data in the presence of normal forms. Below I will give an implementation of this system in Maude. Instead of exploring the search space, by using the `search` function which implements breadth-first search, to find a proof of the given structure, in this implementation proofs are deterministically computed by following the above strategy. This implementation also demonstrates how meta-level features of language Maude can be used to implement a given strategy. Although the associative commutative equational theory is redundant for system DKSg, I prefer to use associativity and commutativity because this allows me to accomplish the implementation by only employing the `reduce` function without employing the `search` function. This way, instead of searching for a proof, it became possible to compute the proof deterministically.

```
fmod DKS-Signature is
  inc META-LEVEL .

  sort Atom .
  sort Structure .
  subsort Atom < Structure .

  ops a b c d e f g h i j k l m n p q r : -> Atom .

  ops tt    : -> Atom .
  ops ff    : -> Atom .
  op -_     : Structure -> Structure [ prec 50 ] .
  op [_,_]  : Structure Structure -> Structure [assoc comm] .
  op {_,_}  : Structure Structure -> Structure [assoc comm] .
```

```
  *** a dummy element for the meta-level 'kind' management
  op dummy : -> [Structure] .
endfm

fmod DKS-NNF is
  inc DKS-Signature .

  var R T U : Structure .

     eq - tt = ff .
     eq - ff = tt .
     eq - [ R , T ] = { - R , - T } .
     eq - { R , T } = [ - R , - T ] .
     eq - - R  = R .
endfm

fmod DKS-distribute is
  inc DKS-Signature .

  var R T U : Structure .

   eq  [ U , { R , T } ]  = { [U,R] , [U,T] } .
endfm

fmod DKS-interaction is
  inc DKS-Signature .

  var A : Atom .

   eq   [ A , - A ]  = tt .
 endfm

fmod DKS-weakening is
  inc DKS-Signature .

  var R : Structure .

   eq   [ tt , R ]  = tt .
   eq   { tt , R }  =  R  .
 endfm

fmod DKS-Strat is
     inc DKS-Signature .
     inc DKS-NNF .
     inc DKS-distribute .
     inc DKS-interaction .
     inc DKS-weakening .
```

```
     op prove_ : Structure -> Structure .

     var R : Structure .

eq prove R =
 downTerm(
  getTerm(
   metaReduce(['DKS-weakening],
    getTerm(
     metaReduce(['DKS-interaction],
      getTerm(
       metaReduce(['DKS-distribute],
        getTerm(metaReduce(['DKS-NNF], upTerm( R )))))))), dummy) .
endfm
```

The language Maude does not allow the passing of a value computed by a module to another one in a straightforward way as it is usually the case in functional or logic programming languages. The way around this problem is to use the `META-LEVEL` module, which provides the means to meta-represent modules and terms so that modules and values that they compute can be treated as syntactic objects inside another module. This is due to the *reflective* features of rewriting logic [**Cla00**]: "a reflective logic is a logic in which important aspects of its meta-theory can be represented at the object level in a consistent way, so that the object-level representation correctly simulates the relevant meta-theoretic aspects." In other words, a reflective logic is a logic which can be faithfully interpreted in itself (see, e.g., [**CDE$^+$99, Cla00, CM96**]). Maude implements these reflective features of rewriting logic by means of the built-in `META-LEVEL` module.

In the implementation above, the different phases of the proof, where different sets of inference rules are used, are represented by functional modules which are called by the operator `prove` of the functional module `DKS-Strat`. Seen procedurally, by means of the operation `upTerm`, this operator first converts the object level representation of the input query term to a Maude meta-level representation of the same term with respect to the module `DKS-Signature`. Then the meta-level term corresponding to the negation normal form of the input term is computed by means of the operation `metaReduce` which takes the meta-representation of the functional module `DKS-NNF` as argument. Then the computed meta-level terms are passed similarly to the meta-level representations of the functional modules `DKS-distribute`, `DKS-interaction` and `DKS-weakening`, respectively, which reduce these meta-level terms with respect to their rules.

In the language Maude it is possible to define a notion of (implicit) error supersorts called *kinds*, which are represented as sort names in square brackets. In the module `DKS-Signature` above, the operator `dummy` belongs to such a kind `Structure`, which is an argument of the meta-level function `downTerm`. The operation `downTerm`, which allows moving from meta-level to object level, then delivers the computed term. If the input structure is a provable KSg structure, this term is the unit `tt`.

Furthermore, as the reader might realize, in the implementation above, the modules `DKS-interaction` and `DKS-weakening` are called in the reversed order

with respect to the order given in Theorem 4.56. This is because these rules are applied on disjunctions in a conjunctive normal form: By exploiting the associative commutative equational system, an atom and its dual can be annihilated in a disjunction. Following this, the rule weakening can be applied to the rest of the atoms in the disjunction. This allows to compute the desired term deterministically without performing a search.

REMARK 4.57. *It is well known that cut-free sequent calculus does not polynomially simulate*[3] *(see, e.g.,* [**BP98**]*) popular proof procedures such as resolution. However, by recomposing the inference rules of system* KSgn*, it is also possible to model different proof procedures while protecting the applicability of the inference rules at any depth inside a structure. For instance the rule*

$$\mathsf{res}\,\frac{S[(R,a,\ldots,a),(T,\bar{a},\ldots,\bar{a}),(R,T)]}{S[(R,a,\ldots,a),(T,\bar{a},\ldots,\bar{a})]}\ ,$$

which is derivable in KSg*, is the (proof) resolution rule (see, e.g.,* [**Bus98**]*). The dual (contrapositive) rule of this rule is the refutation resolution rule.*

4.4. Discussion

In this chapter, we have seen implementations of the systems of the calculus of structures in Maude. The language Maude supports implementing term rewriting systems modulo different combinations of associative commutative equational systems, also in the presence of units. Maude has a simple high level language, and a built-in search function, which implements breadth-first search. These features of Maude make it an appropriate language for implementing the systems of the calculus of structures. The syntax of these systems and their inference rules can be easily expressed in Maude and a complete search strategy can be effectively employed. This way, it becomes possible to observe a one-to-one correspondence between the proof theoretical systems and the Maude modules that implement these systems, and to consider these systems as executable, computation as proof construction tools for these logics. Furthermore, despite being rather complex, meta-level features of Maude are useful for implementing different search strategies, especially when intermediate normal forms during proof search are available.

As we have seen in this chapter, equational systems of the systems NEL and LS require special treatment in Maude: The equational systems of NEL and LS include equations for the exponentials, which cannot be expressed as Maude operator attributes, unlike those for associativity, commutativity, and units. In the equivalent systems that I presented in this chapter, the equations for exponentials become redundant, thus these resulting systems can be implemented in Maude.

The equations for units often cause redundant matchings of the inference rules, resulting in trivial instances of the inference rules: The premise and the conclusion of these rule instances are equivalent structures. In order to avoid such instances, in an automated rule application of an inference rule, I presented equivalent systems, where the role played by the equations for units is made explicit in the inference rules. The resulting systems do not only get rid of the trivial instances of the inference rules, but also provide shorter proofs, and this way provide a better performance in proof search.

[3]A proof system U polynomially simulates a proof system V if both U and V prove the same language and proofs in V can be converted to into proofs in V in polynomial time.

In this chapter, I discussed the systems BV, NEL, LS, and KSg, considering only their down-fragments. However, these methods can be analogously applied to the other systems of the calculus of structures. Further, the rules belonging to the up-fragment of a system can be freely added to a Maude module.

The modules presented in this chapter consider the proof theoretical systems from an analytical bottom-up proof construction point of view. This points out the potential applications of these systems as logic programming languages, or directly implementable operational semantics. In Chapter 8, I give an example of such a usage.

An aspect that distinguishes my implementation of system LS, and also system NEL (which cannot be designed in the sequent calculus,) is due to the *promotion* rule. The Maude implementation of a sequent calculus system for linear logic in [**MOM96**], involves this rule which requires a global knowledge of the context. In the calculus of structures, thus in the implementations presented in this chapter, this rule is replaced with a *local* rule which does not require such a global view of the formulae. The implementations of systems BV and NEL, presented here, are the first implementations of these systems. All the implementations discussed in this chapter are available online [4].

The implementations presented in this chapter are implemented in language Maude, so they are run on the command-line prompt of this language. Because of this, using these implementations requires some knowledge of the language Maude. Furthermore, the output generated by these tools is somewhat remote from the usual presentation of the calculus of structures derivations. Schäfer has developed a graphical proof editor, called GraPE [**Sch06**], which functions as a user-friendly graphical user interface to these Maude modules and makes it possible to use the Maude implementations presented in this chapter interactively: By using the GraPE tool, the user can guide the proof construction and choose between automated proof search and user-guided proof construction. Then the output derivation can be exported as LaTeXcode. The GraPE tool is available online [5].

[4] `http://www.iccl.tu-dresden.de/~ozan/maude_cos.html`

[5] `http://grape.sourceforge.net/grape.pdf`

CHAPTER 5

Reducing Nondeterminism in Proof Search

The deep inference feature of the calculus of structures does not only provide a richer combinatorial analysis of the logic being studied, but also provides shorter proofs than any other formalism supporting analytical proofs (see, e.g., [**Gug04c**]): As we have seen on an example in Chapter 1, applicability of the inference rules at any depth inside a structure makes it possible to start the construction of a proof by manipulating and annihilating substructures. However, deep inference causes a greater nondeterminism in proof search: Because the inference rules can be applied at many more positions, the breadth of the search space increases rather quickly. Let us see this on the following examples.

EXAMPLE 5.1. *To the structure* $[(\bar{a},\bar{b}),a,b]$ *the switch rule in system* BV *can be applied bottom-up in 12 different ways that are shown below. (In system* KSg*, the switch rule can be applied to this structure in 27 different ways.) The instances* (1.) *to* (6.) *are the instances which result from the applications of equations for unit. The trivial instances, such as those in Example 2.13, are excluded below. From all the 12 instances, only two of these instances can lead to a proof, namely* (8.) *and* (10.) *below:*

$$(1.)\ \mathsf{s}\frac{(a,[b,(\bar{a},\bar{b})])}{[(\bar{a},\bar{b}),a,b]} \qquad (2.)\ \mathsf{s}\frac{(b,[a,(\bar{a},\bar{b})])}{[(\bar{a},\bar{b}),a,b]} \qquad (3.)\ \mathsf{s}\frac{(\bar{a},\bar{b},[a,b])}{[(\bar{a},\bar{b}),a,b]}$$

$$(4.)\ \mathsf{s}\frac{[b,(a,\bar{a},\bar{b})]}{[(\bar{a},\bar{b}),a,b]} \qquad (5.)\ \mathsf{s}\frac{[(a,b),(\bar{a},\bar{b})]}{[(\bar{a},\bar{b}),a,b]} \qquad (6.)\ \mathsf{s}\frac{[a,(b,\bar{a},\bar{b})]}{[(\bar{a},\bar{b}),a,b]}$$

$$(7.)\ \mathsf{s}\frac{[a,(\bar{b},[b,\bar{a}])]}{[(\bar{a},\bar{b}),a,b]} \qquad (8.)\ \mathsf{s}\frac{[a,(\bar{a},[b,\bar{b}])]}{[(\bar{a},\bar{b}),a,b]} \qquad (9.)\ \mathsf{s}\frac{(\bar{b},[a,b,\bar{a}])}{[(\bar{a},\bar{b}),a,b]}$$

$$(10.)\ \mathsf{s}\frac{[b,(\bar{b},[a,\bar{a}])]}{[(\bar{a},\bar{b}),a,b]} \qquad (11.)\ \mathsf{s}\frac{[b,(\bar{a},[a,\bar{b}])]}{[(\bar{a},\bar{b}),a,b]} \qquad (12.)\ \mathsf{s}\frac{(\bar{a},[a,b,\bar{b}])}{[(\bar{a},\bar{b}),a,b]}$$

In particular, when only system FBV *is considered, there are altogether 358 derivations in the proof-search space of the structure* $[(\bar{a},\bar{b}),a,b]$*, however only 6 of these derivations are proofs.*

In the example above, none of the rule instances is deep. However, one can observe the redundant nondeterminism in these instances. The availability of deep inference causes an even greater redundant nondeterminism in the structures where the inference rules can be applied at any depth inside structures.

EXAMPLE 5.2. *Consider the structure below which is obtained by nesting the structure above in itself:*

$$[([\bar{a}_1,(\bar{a}_2,\bar{b}_2),a_2,b_2],\bar{b}_1),a_1,b_1]$$

To this structure, the switch rule can be applied in 51 different ways but only 4 of these instances can lead to a proof.

The process of searching for a proof within a certain deductive system is a nondeterministic process (of course, if the subject logic is not in P, for instance, like propositional Horn logic which is linear). From the point of view of logic programming, this is a central feature which is due to the nature of logic. For instance, given that multiplicative linear logic (MLL) is NP-complete [**Kan91**], if P is different than NP, as many believe, coming up with a tractable algorithm for MLL is not feasible. However, in proof search usually not all nondeterminism is meaningful: In a proof search episode there are often some nondeterministic choices to be made, i.e., deciding which rule instances to apply. In the process of searching for a proof some of these actions must be avoided, because they result in a dead-end and require backtracking, whereas others must be taken sooner or later so that a proof can be constructed.

Reducing nondeterminism in proof search without losing the completeness of the subject system requires combinatorial techniques that work in harmony with the proof theoretical formalism. Because the rules of the sequent calculus act on the main connective, and the notion of main connective resolves in the systems with deep inference, it is impossible to use the techniques of the sequent calculus. For instance, Andreoli's focusing technique [**And92, And01**], which was introduced to attack this problem within linear logic in the sequent calculus, exploits the applications of the inference rules at the main connective: The focusing technique is based on permuting different phases of a proof by distinguishing between asynchronous (deterministic) and synchronous (nondeterministic) parts of a proof. This approach depends on the fact that in the sequent calculus asynchronous connectives, e.g., par, and synchronous connectives, e.g., copar, can be treated in isolation. However, in the calculus of structures connectives are never in isolation: Asynchronous connectives are always matched to a synchronous connective at an inference step. Furthermore, asynchronous parts of a proof map the object level, given by the logical operators, onto the meta-level. For instance, par operators are mapped to commas. In the systems with deep inference this is a superfluous operation, because what is meta-level in the sequent calculus is brought to the object level, thus there is no meta-level.

Another source of nondeterminism in the sequent calculus is due to the inference rules that are responsible for the context management. For instance, let us consider the sequent calculus $\otimes$ rule which splits the context of the main formula:

$$\otimes \frac{\vdash \Gamma\,,\, A \quad \vdash \Sigma\,,\, B}{\vdash \Gamma\,,\, \Sigma\,,\, A \otimes B}$$

In a bottom-up proof construction, if the multiset contexts of a sequent have $n \geq 0$ formulas in it, then there can be as many as 2^n ways that a context is partitioned into two multisets. However, often very few of these splits will lead to a successful proof. Application of this rule results in a decision that binds Σ with the right (or left) branch of the proof tree, thereby making the communication of A with Σ possible only by backtracking. This is an exponential source of unwanted nondeterminism.

The nondeterminism due to the context management rules in the sequent calculus implementations was previously addressed by various authors: Hodas and

Miller propose an approach, in [**HM94**], for splitting the context lazily by observing proof search as a kind of input/output process: When one part of a tensor is being proved all of the formulas in the context are given to that part. Due the branching in the sequent calculus proof, this will be the first branch in case of a successful attempt, and then the rest of the context which is not consumed by the first branch is given to the other part of the tensor. [**HM94**] includes a description of a simple Prolog interpreter of this approach. Several researchers have developed variations to the lazy splitting approach in the sequent calculus implementations (see, e.g., [**CHP96, Hod94**]). Some other later approaches for context management in linear logic programming use constraint solving techniques, also addressing resource consumption issues which arise at the different parts of a sequent calculus proof (see, e.g., [**And01, HP03**]).

In the calculus of structures, the inference rule that is responsible for the (commutative) context management is the switch rule. Although the switch rule manages the context in a lazier way and breaks the interaction between structures rather gradually, this problem persists as it can be observed in the above examples. Moreover, applicability of the inference rules at any depth inside structures introduces further nondeterminism which is not present in the sequent calculus.

In this chapter, I will introduce a proof theoretical technique in the calculus of structures that reduces nondeterminism in proof search. This technique exploits an intuitive observation on the mutual relations between atoms of the structure being proved. These mutual relations are those of the graphical representations of structures, called *relation webs*. Observed from the point of view of such relations between atoms, the duty of the inference rules can be seen as starting from a set of pairs of interacting atoms, reducing the interaction between atoms of the structure, and finally arriving at a set of pairs of interacting atoms which are dual atoms, having the same interaction with the other atoms in the structure. In other words, proofs are constructed by promoting the interaction in the sense of a specific mutual relation between dual atoms, and annihilating these dual atoms while going up in a derivation. This technique also makes the shorter proofs, that are available due to deep inference, more immediately accessible.

In the following, I will present the relation webs which are helpful to develop and understand the ideas that I will present later in this chapter. I will then present a class of systems equivalent to system BV, where nondeterminism in proof search is reduced at different levels by using this technique. I will present experimental results that demonstrate the performance improvement in a Maude implementation. This technique exploits the common scheme, which is obeyed by all the systems of the calculus of structures, and generalizes to all these other systems. As an evidence to this, I will present a system equivalent to system KSg where nondeterminism is reduced by this technique.

5.1. Relation Webs

In this section, we will see a characterization of the BV structures by means of special graphs, called *relation webs*. Relation webs were introduced in [**Gug07**], where Guglielmi uses them to derive the inference rules of system BV by asking for a certain conservation property to hold while manipulating the structures, thus their relation webs. In [**Tiu01**], Tiu uses the relation webs to show that deep inference is essential for a deductive system to get all the provable structures of system BV.

Relation webs are helpful to observe the mutual relations between atoms in a structure with respect to the logical operators with which these atoms are related. Because an application of the inference rules manipulates these mutual relations in a specific way, they will be useful to make observations about the role played by the inference rules during the construction of a proof. In the later sections, I will exploit these observations while redesigning the inference rules. Thus, they play a key role in the development of these ideas. Relation webs are fully developed for system BV, however the intuitions behind them apply also to other systems.

A relation web is a complete graph whose nodes are all atom occurrences of a structure. Relation webs can be considered as canonical graph representations of equivalence classes of structures, that is, there is a unique relation web for every equivalence class of structures:

DEFINITION 5.3. *Given a structure R, $\mathsf{at}\,R$ is the set of all the atoms appearing in R.*

DEFINITION 5.4. *We talk about* atom occurrences *when considering all the atoms appearing in R as distinct (for example, by indexing them so that two atoms which are equal get different indices). Given a structure R, $\mathsf{occ}\,R$ is the set of all the atom occurrences appearing in R. The* size *of R is the cardinality of the set $\mathsf{occ}\,R$. Let R be a structure in unit normal form. The four* structural relations $\lhd_R$ (seq), $\rhd_R$ (aseq), $\downarrow_R$ (par), *and* $\uparrow_R$ (co-par) *are defined as the minimal sets such that*

$$\lhd_R, \rhd_R, \downarrow_R, \uparrow_R \subset (\mathsf{occ}\,R) \times (\mathsf{occ}\,R)$$

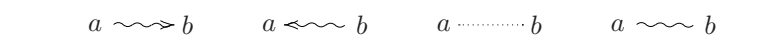
and for every $S\{\ \}$, U and V and for every atom occurrence a in U and b in V the following holds:

- *if $R = S\langle U; V\rangle$ then $a \lhd_R b$ and $b \rhd_R a$;*
- *if $R = S[U, V]$ then $a \downarrow_R b$;*
- *if $R = S(U, V)$ then $a \uparrow_R b$.*

To a structure that is not in unit normal form we associate the structural relation obtained from any of its normal forms, because they yield the same relation $\downarrow_R$. The quadruple $(\mathsf{occ}\,R, \lhd_R, \downarrow_R, \uparrow_R)$ is called the relation web *of R, it is denoted by w_R. We shall omit the subscripts in $\lhd_R$, $\rhd_R$, $\downarrow_R$, and $\uparrow_R$, if it is clear from context which structure we refer to. Given two sets of atom occurrences μ and ν, we write $\mu \downarrow \nu$ to denote that, for every a in μ and for every b in ν, it holds that $a \downarrow b$. The notation $|\downarrow_R|$ denotes the cardinality of the set $\downarrow_R$.*

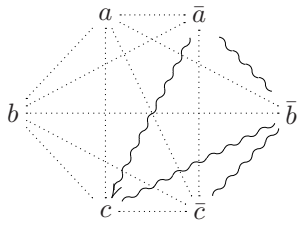
EXAMPLE 5.5. *In order to see the above definition at work, consider the following structure: $R = [a, b, (\bar{b}, [\langle\bar{a}; c\rangle, \bar{c}])]$. We have $\mathsf{at}\,R = \mathsf{occ}\,R = \{a, \bar{a}, b, \bar{b}, c, \bar{c}\}$. Then in w_R, we have $a \downarrow b$, $a \downarrow \bar{b}$, $a \downarrow \bar{a}$, $a \downarrow c$, $a \downarrow \bar{c}$, $b \downarrow \bar{b}$, $b \downarrow \bar{a}$, $b \downarrow c$, $b \downarrow \bar{c}$, $\bar{a} \downarrow \bar{c}$, $c \downarrow \bar{c}$, $\bar{a} \uparrow \bar{b}$, $c \uparrow \bar{b}$, $\bar{c} \uparrow \bar{b}$, $\bar{a} \lhd c$, (we omit the symmetric relations, e.g., $b \downarrow a$).*

Structural relations between occurrences of atoms are represented by drawing



when $a \lhd b$, $b \rhd a$, $a \downarrow b$ and $a \uparrow b$, respectively.

EXAMPLE 5.6. *Let us now consider the structure* $R = [a, b, (\bar{b}, [\langle \bar{a}; c\rangle, \bar{c}])]$ *graphically as a relation web:*



Guglielmi proves the following result in [**Gug07**].

THEOREM 5.7. *Two* BV *structures are equivalent if and only if they have the same relation web.*

Intuitively, one can consider the relation $\downarrow_R$ as a notion of interaction, and the relations $\uparrow_R$ and $\lhd_R$ ($\rhd_R$) as non-interaction. In other words, the atoms which are related by $\downarrow_R$ are interacting atoms, whereas others are non-interacting: Proofs are constructed by isolating the interacting atoms in a way such that each atom preserves the interaction with a dual in the relation web. Such interacting dual atoms are annihilated at an application of the atomic interaction rule when they share the same interaction/non-interaction scheme with the rest of the atoms in the relation web. During a bottom-up proof search episode, while acting on structures, inference rules perform such an isolation of atoms: In an instance of an inference rule with the conclusion R, the inference rules transform some structural relations $\downarrow_R$ into structural relations $\uparrow_R$ and $\lhd_R$ ($\rhd_R$), at applications of the switch and seq rules, respectively, until dual atoms establish the same structural relations with all the other atoms. Then an atomic interaction rule can be applied to an atom and its dual, in a $\downarrow_R$ structural relation, when both of these atoms have the same set of structural relations with all the other atoms of the structure. Figure 5.1 demonstrates the role of the inference rules from the point of view of relation webs.

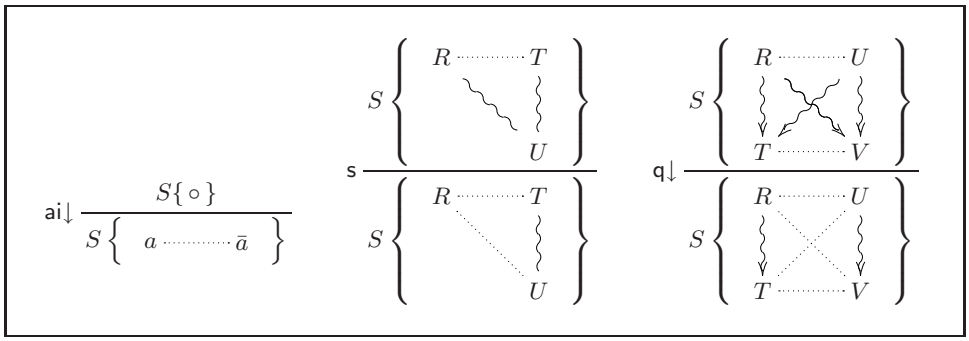
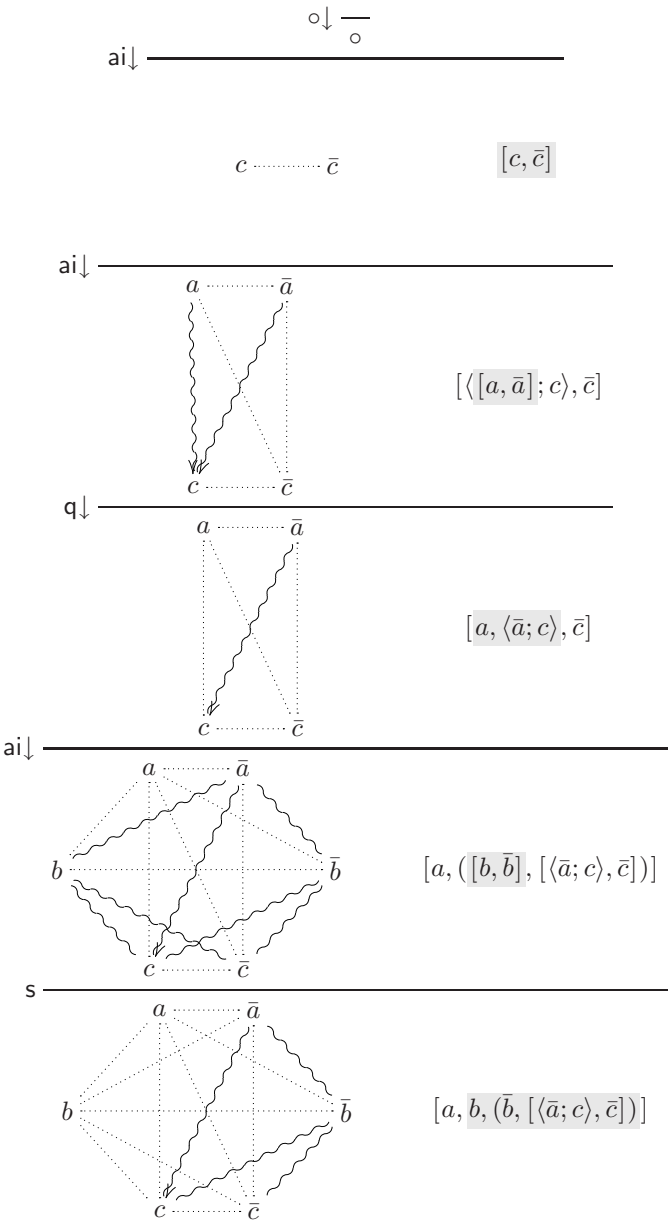


FIGURE 5.1. The relation web view of the inference rules of system BV

EXAMPLE 5.8. *Let us now see a proof of the structure* $[a, b, (\bar{b}, [\langle \bar{a}; c\rangle, \bar{c}])]$ *and the relation webs corresponding to the structures resulting from the instance of the inference rules at each step of the proof:*



Often inference rules can be applied to a structure in many different ways, however only few of these applications can lead to a proof, for instance as in Example 5.1.

PROPOSITION 5.9. *If a structure R has a proof in* BV *then, for all the atoms a that appear in R, there is an atom $\bar{a}$ in R such that $a \downarrow_R \bar{a}$.*

PROOF. With induction on the length k of the proof Π of R. The base case is given by the proof consisting of the application of the rule $\circ\downarrow$ which has no atoms in it. Moving to the induction step, we assume that the proposition holds for proof

of length k. Let us construct the structure R such that ρ is the bottom most rule instance in the proof Π with length $k+1$ and R, and R' are, respectively, the premise and the conclusion of ρ. With the induction hypothesis, for all the atoms a in R', there is an atom $\bar{a}$ such that $a \downarrow \bar{a} \in \mathsf{w}_{R'}$. If ρ is $\mathsf{ai}{\downarrow}$, then for $R' = S\{\circ\}$ the proof is trivial. If ρ is

- s, then, for $R' = S([P\{a\}, T\{\bar{a}\}], U)$ take
$$\mathsf{s}\frac{S([P\{a\}, T\{\bar{a}\}], U)}{S[(P\{a\}, U), T\{\bar{a}\}]} \quad ;$$
- $\mathsf{q}{\downarrow}$, then for $R' = S\langle [P\{a\}, T\{\bar{a}\}]; [U, V]\rangle$ take
$$\mathsf{q}{\downarrow}\frac{S\langle [P\{a\}, T\{\bar{a}\}]; [U, V]\rangle}{S[\langle P\{a\}; U\rangle, \langle T\{\bar{a}\}; V\rangle]} \quad .$$

□

The following example is helpful to understand the intuition behind the proposition above.

Example 5.10. *Consider the 12 derivations in Example 5.1. While going up in these derivations, from conclusion to premise the following structural relations cease to hold: in* (1.)*, $a \downarrow b$, $a \downarrow \bar{a}$ and $a \downarrow \bar{b}$; in* (2.)*, $b \downarrow a$, $b \downarrow \bar{a}$ and $b \downarrow \bar{b}$; in* (3.) *$a \downarrow \bar{a}$ and $b \downarrow \bar{b}$; in* (4.) *$a \downarrow \bar{b}$ and $b \downarrow \bar{b}$; in* (5.) *$a \downarrow \bar{a}$ and $b \downarrow \bar{a}$; in* (6.) *$b \downarrow \bar{a}$ and $b \downarrow \bar{b}$; in* (7.) *$b \downarrow \bar{b}$; in* (8.) *$b \downarrow \bar{a}$; in* (9.) *$a \downarrow \bar{a}$ and $a \downarrow \bar{b}$; in* (10.) *$a \downarrow b$; in* (11.) *$a \downarrow \bar{a}$; and in* (12.) *$a \downarrow b$. Only the instances* (8.) *and* (10.) *provide proofs.*

It is easy to see that a structure is not necessarily provable if for every atom a and its dual $\bar{a}$ we have that $a \downarrow \bar{a}$. For instance, the structure $[(a, \bar{a}), (a, \bar{a})]$ is not provable.

Let me now state the following remarks which follow from the discussions in this section.

Remark 5.11. *Let $R = S[a, \bar{a}]$ and $R' = S\{\circ\}$ be* BV *structures. If $\mathsf{ai}{\downarrow}\dfrac{R'}{R}$, then $\downarrow_{R'} = \downarrow_R \setminus \{(a, \bar{a}), (\bar{a}, a)\}$.*

Remark 5.12. *Let $R = S[(P, T), U]$ and $R' = S([P, U], T)$ be* BV *structures. If $\mathsf{s}\dfrac{R'}{R}$, then*

$$\downarrow_{R'} = \downarrow_R \setminus (\{(x, y) \mid x \in \mathsf{occ}\, T \wedge y \in \mathsf{occ}\, U\} \cup \{(x, y) \mid x \in \mathsf{occ}\, U \wedge y \in \mathsf{occ}\, T\}).$$

Remark 5.13. *Let $R = S[\langle P; T\rangle, \langle U; V\rangle]$ and $R' = S\langle [P, U]; [T, V]\rangle$ be* BV *structures. If $\mathsf{q}{\downarrow}\dfrac{R'}{R}$, then*

$$\begin{aligned}\downarrow_{R'} = \downarrow_R \setminus \quad & (\{(x, y) \mid x \in \mathsf{occ}\, P \wedge y \in \mathsf{occ}\, V\} \cup \{(x, y) \mid x \in \mathsf{occ}\, V \wedge y \in \mathsf{occ}\, P\} \cup \\ & \{(x, y) \mid x \in \mathsf{occ}\, U \wedge y \in \mathsf{occ}\, T\} \cup \{(x, y) \mid x \in \mathsf{occ}\, T \wedge y \in \mathsf{occ}\, U\}).\end{aligned}$$

We can now state the following proposition.

PROPOSITION 5.14. *The length of a proof of a* BV *structure* R *is bounded by* $\mathcal{O}(|\mathsf{occ}\, R|^2)$.

PROOF. With Remark 5.11, 5.12, and 5.13; observe that $\downarrow_R \subseteq (\mathsf{occ}R)\times(\mathsf{occ}R)$, hence $|\downarrow_R| \leq |\mathsf{occ}R|^2$. For each (non-trivial) application of an inference rule such that $\rho\dfrac{R'}{R}$, we have that $|\downarrow_{R'}| < |\downarrow_R|$. □

In the next sections, I will exploit these observations to redesign the inference rules of system BV such that some instances of the inference rules that cannot provide a proof of a provable structure will be disabled.

5.2. The Switch Rule

As we have seen in Example 5.1, in a proof search episode the applicability of the inference rules, in particular the switch rule, in different ways causes a redundant nondeterminism. With the below definition, I will re-design the switch rule such that only those applications, which are meaningful from the point of view of proof search, will be possible.

DEFINITION 5.15. *Let* interaction switch *be the rule*

$$\mathsf{is}\,\frac{S([R,W],T)}{S[(R,T),W]}$$

where $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,R \neq \emptyset$.

DEFINITION 5.16. *The rule* lazy interaction switch*, or* lis*, is the instance of the interaction switch rule where the structure* W *is not a proper par structure.*

DEFINITION 5.17. System BV with interaction switch*, or system* BVs*, is the system* $\{\circ\downarrow, \mathsf{ai}\downarrow, \mathsf{is}, \mathsf{q}\downarrow\}$.

DEFINITION 5.18. System BV with lazy interaction switch*, or system* BVsl*, is the system resulting from replacing the rule* is *in* BVs *with the rule* lis.

EXAMPLE 5.19. *It is important to observe that the rule* lis *can be applied bottom-up to the structure* $[(\bar{a},\bar{b}),a,b]$ *only as in the cases* (8.) *and* (10.) *of Example 5.1.*

$$\mathsf{lis}\,\frac{[([\bar{a},a],\bar{b}),b]}{[(\bar{a},\bar{b}),a,b]} \qquad\qquad \mathsf{lis}\,\frac{[([\bar{b},b],\bar{a}),a]}{[(\bar{a},\bar{b}),a,b]}$$

The switch rule can be safely replaced with the lazy interaction switch rule in system BV without losing completeness. In the following, I will collect some definitions and lemmas that will be necessary to prove this result.

PROPOSITION 5.20. *Let* $\mathscr{S} \in \{\mathsf{BV}, \mathsf{BVs}, \mathsf{BVsl}\}$. *In system* $\mathscr{S}$

(*i*) $\langle R;T\rangle$ *is provable if and only if* R *and* T *are provable;*
(*ii*) (R,T) *is provable if and only if* R *and* T *are provable.*

PROOF. The only if direction is trivial. For the proof of the if direction, the proof for the (*ii*) being analogous, let us see the proof for case (*i*). With induction on the length of proof Π of $\langle R;T\rangle$, we construct proofs of R and T: The base case is trivial. Returning to the inductive cases, we do case analysis on the last rule ρ applied in Π. The redex of ρ must be inside either in R or T, because otherwise the

rule $\rho \in \mathscr{S}$ cannot be applied to $\langle R; T\rangle$. The case where the redex inside T being analogous, the case where the redex is inside R is the proof

$$\rho\dfrac{\begin{array}{c}\Pi\Big\|\mathscr{S}\\ \langle R';T\rangle\end{array}}{\langle R;T\rangle}\quad.\qquad\text{Take the proofs}\qquad\begin{array}{c}\Pi_1\Big\|\mathscr{S}\\ \rho\dfrac{R'}{R}\end{array}\qquad\text{and}\qquad\begin{array}{c}\Pi_2\Big\|\mathscr{S}\\ T\end{array}$$

where the proofs Π_1 and Π_2 are delivered by the induction hypothesis. □

Definition 5.21. *Let R, T be* BV *structures such that $R \neq \circ \neq T$ and let $\mathscr{S} \in \{\mathsf{BV}, \mathsf{BVs}, \mathsf{BVsl}\}$. R and T are* independent (for $\mathscr{S}$) *if and only if*

$$\begin{array}{c}\Big\|\mathscr{S}\\ [R,T]\end{array}\qquad\textit{implies}\qquad\begin{array}{c}\Big\|\mathscr{S}\\ R\end{array}\qquad\textit{and}\qquad\begin{array}{c}\Big\|\mathscr{S}\\ T\end{array}\quad.$$

Otherwise, they are dependent.

Example 5.22. *For the structure $S = [a, b, (\bar{a}, \bar{b}), \langle [c, \bar{c}]; [a, \bar{a}]\rangle]$,*

$$R = [a, b, (\bar{a}, \bar{b})] \quad \textit{and} \quad T = \langle [c, \bar{c}]; [a, \bar{a}]\rangle$$

are independent, whereas

$$R' = [a, b] \quad \textit{and} \quad T' = [(\bar{a}, \bar{b}), \langle [c, \bar{c}]; [a, \bar{a}]\rangle]$$

are dependent.

Proposition 5.23. *For any* BV *structures R and T, if $\mathsf{at}\,\bar{R} \cap \mathsf{at}\,T = \emptyset$ then R and T are independent.*

Proof. Assume that there is a proof Π of $[R, T]$. Construct a proof of R by replacing all the substructures of T in Π with $\circ$: All the instances of the rules s and $\mathsf{q}{\downarrow}$ remain intact. Further, from Proposition 5.9 it follows that all the instances of the rule $\mathsf{ai}{\downarrow}$ remain intact, because for every atom $a \in \mathsf{at}\,[R, T]$ there must be an atom $\bar{a} \in \mathsf{at}\,[R, T]$ and we have that $\mathsf{at}\,\bar{R} \cap \mathsf{at}\,T = \emptyset$. This implies that each instance of the rule $\mathsf{ai}{\downarrow}$ in Π annihilates an atom and its dual that are both either in $\mathsf{at}\,R$ or in $\mathsf{at}\,T$. Analogously, construct a proof of T by replacing all the substructures of R in Π with $\circ$. □

Lemma 5.24. *For any* BV *structures P, U, and R,*

$$\textit{if}\qquad\begin{array}{c}\Pi\Big\|\mathsf{BVsl}\\ [P,U]\end{array}\qquad\textit{then there is a derivation}\qquad\begin{array}{c}R\\ \Big\|\mathsf{BVsl}\\ [(R,P),U]\end{array}\quad.$$

Proof. We label each atom occuring in Π such that every pair of atom that is annihilated by an application of the rule $\mathsf{ai}{\downarrow}$ get the same label, and the conclusion of each rule instance in Π consists of pairwise distinct atoms. If U is a proper, then there must be U_1 and U_2 such that $U = [U_1, U_2]$, and $\mathsf{at}\,[P, U_1] \cap \mathsf{at}\,\overline{U_2} = \emptyset$. If U is not a proper par then it must be that either $U_1 = U$ and $U_2 = \circ$ or $U_1 = \circ$ and $U_2 = U$. Thus, there is a derivation

$$\begin{array}{c}[(R,[P,U_1]),U_2]\\ \Delta\Big\|\{\mathsf{lis}\}\\ [(R,P),U]\end{array}\quad.$$

Given that $[P, U_1, U_2]$ is provable, from Proposition 5.23, it follows that $[P, U_1]$ and U_2 are independent, which implies that there are proofs $\begin{array}{c}\Pi_1 \\ \hline\!\!\hline \mathsf{BVsl} \\ [P, U_1]\end{array}$ and $\begin{array}{c}\Pi_2 \\ \hline\!\!\hline \mathsf{BVsl} \\ U_2\end{array}$. We can then construct the following derivation:

$$\begin{array}{c} R \\ \Pi_2 \,\big\|\, \mathsf{BVsl} \\ [R, U_2] \\ \Pi_1 \,\big\|\, \mathsf{BVsl} \\ [(R, [P, U_1]), U_2] \\ \Delta \,[\!\big|\, \mathsf{BVsl} \\ [(R, P), U_1, U_2] \end{array}$$

□

The following theorem is a specialization of the shallow splitting theorem which was introduced, in [**Gug07**], for proving cut elimination for system BV. In the following, I will use this theorem to show the completeness of system BVsl. By exploiting the fact that systems in the calculus of structures follow a scheme, in which the rules atomic interaction and switch are common to all other systems, this technique was used also to prove cut elimination for classical logic [**Brü03b, Gug04d**], linear logic [**Str03a**], and system NEL [**GS02, Str03c**]. As the name suggests, this theorem splits the context of a structure so that the proof of the structure can be partitioned into smaller pieces in a systematic way. Below we will see that the splitting theorem can be specialized to system BVsl.

It is possible to prove the theorem below by following exactly the same scheme as in [**Gug07**]. However, in the below proof I use a one-dimensional induction measure, in contrast to Gugliemi's two dimensional induction measure. This results in a simpler proof:

THEOREM 5.25. *(Shallow splitting for* BVsl*) For all structures R, T and P:*

(1) *if* $[\langle R; T\rangle, P]$ *is provable in* BVsl *then there exists* P_1, P_2 *and* $\begin{array}{c}\langle P_1; P_2\rangle \\ \Delta \,\big\|\, \mathsf{BVsl} \\ P\end{array}$ *such that* $[R, P_1]$ *and* $[T, P_2]$ *are provable in* BVsl.

(2) *if* $[(R, T), P]$ *is provable in* BVsl *then there exists* P_1, P_2 *and* $\begin{array}{c}[P_1, P_2] \\ \Delta \,\big\|\, \mathsf{BVsl} \\ P\end{array}$ *such that* $[R, P_1]$ *and* $[T, P_2]$ *are provable in* BVsl.

PROOF. All the derivations below are in BVsl. Consider the following two statements:

$$\begin{aligned}\mathsf{S}(n) = \forall n'.\forall R,T,P.\Big((n' \leq n \\ \wedge \quad n' = |\downarrow_{[\langle R;T\rangle,P]}| \\ \wedge \quad \text{there is a proof } \vdots\ [\langle R;T\rangle,P] \\ \Rightarrow \exists P_1,P_2.(\ \begin{matrix}\langle P_1;P_2\rangle \\ \| \\ P\end{matrix} \wedge \begin{matrix}\vdots \\ [R,P_1]\end{matrix} \wedge \begin{matrix}\vdots \\ [T,P_2]\end{matrix})\Big)\ ,\end{aligned}$$

$$\begin{aligned}\mathsf{C}(n) = \forall n'.\forall R,T,P.\Big((n' \leq n \\ \wedge \quad n' = |\downarrow_{[(R,T),P]}| \\ \wedge \quad \text{there is a proof } \vdots\ [(R,T),P] \\ \Rightarrow \exists P_1,P_2.(\ \begin{matrix}[P_1,P_2] \\ \| \\ P\end{matrix} \wedge \begin{matrix}\vdots \\ [R,P_1]\end{matrix} \wedge \begin{matrix}\vdots \\ [T,P_2]\end{matrix})\Big)\ .\end{aligned}$$

The statement of the theorem is equivalent to $\forall n.(\mathsf{S}(n) \wedge \mathsf{C}(n))$ where n is a measure of $(\mathsf{S}(n) \wedge \mathsf{C}(n))$, and the proof is an induction on this measure. The base case is trivial. Let us see the inductive cases. I assume that $P \neq \circ$, because when $P = \circ$ the theorem is trivially proved by Proposition 5.20. Similarly, I assume $R \neq \circ \neq T$. Below, the statements $\mathsf{S}(n)$ and $\mathsf{C}(n)$ will be proved separately.

(1) $\forall n'.(n' < n \wedge \mathsf{S}(n') \wedge \mathsf{C}(n')) \Rightarrow \mathsf{S}(n)$, that is, $|\downarrow_{[\langle R;T\rangle,P]}| = n$ and $[\langle R;T\rangle,P]$ has a proof. Consider the bottom rule instance in this proof:

$$\rho\,\frac{\begin{matrix}\vdots \\ Q\end{matrix}}{[\langle R;T\rangle,P]}\quad ,$$

where I assume that ρ is non-trivial, because every proof with trivial rule instances can be rewritten as a proof where these trivial instances are removed. Let us do case analysis on the position of the redex of ρ in $[\langle R;T\rangle,P]$. We have the following possibilities:

(a) $\rho = \mathsf{ai}{\downarrow}$: The following cases exhaust the possibilities:

(i) The redex is inside R:

$$\text{Given}\qquad \mathsf{ai}{\downarrow}\,\frac{\begin{matrix}\vdots \\ [\langle R';T\rangle,P]\end{matrix}}{[\langle R;T\rangle,P]}\quad ,\quad \text{consider}\qquad \mathsf{ai}{\downarrow}\,\frac{\begin{matrix}\vdots \\ [R',P_1]\end{matrix}}{[R,P_1]}\quad .$$

(ii) The redex is inside T: Analogous to the previous case.

(iii) The redex is inside P:

$$\text{Given} \quad \mathsf{ai}{\downarrow}\,\dfrac{\vdots\,\pi}{[\langle R;T\rangle,P']}}{[\langle R;T\rangle,P]} \quad , \quad \text{consider} \quad \mathsf{ai}{\downarrow}\,\dfrac{\langle P_1;P_2\rangle \,\|\, P'}{P} \quad .$$

(b) $\rho = \mathsf{q}{\downarrow}$: If the redex is inside R, T or P, the situation is analogous to the ones seen above. The following cases exhaust the other possibilities:

(i) $R = \langle R';R''\rangle\,,\ P = [\langle P';P''\rangle, U]$ and

$$\mathsf{q}{\downarrow}\,\dfrac{\vdots\,\pi}{[\langle [R',P'];[\langle R'';T\rangle,P'']\rangle,U]}}{[\langle R';R'';T\rangle,\langle P';P''\rangle,U]} \quad .$$

We can apply the induction hypothesis, by Remark 5.13, and we get

$$\begin{matrix}\langle U_1;U_2\rangle\\ {\scriptstyle\Delta_1}\|\\ U\end{matrix} \quad , \quad \begin{matrix}\pi_1\\ \vdots\\ [R',P',U_1]\end{matrix} \quad \text{and} \quad \begin{matrix}\pi_2\\ \vdots\\ [\langle R'';T\rangle,P'',U_2]\end{matrix} \quad .$$

Because $|\downarrow_{[\langle R'';T\rangle,P'',U_2]}| < |\downarrow_{[\langle R';R'';T\rangle,\langle P';P''\rangle,U]}|$ (otherwise the $\mathsf{q}{\downarrow}$ instance would be trivial), we can apply the induction hypothesis on Π_2, by Remark 5.13, and get

$$\begin{matrix}\langle P_1';P_2\rangle\\ {\scriptstyle\Delta_2}\|\\ [P'',U_2]\end{matrix} \quad , \quad \begin{matrix}\pi_3\\ \vdots\\ [R'',P_1']\end{matrix} \quad \text{and} \quad \begin{matrix}\pi_4\\ \vdots\\ [T,P_2]\end{matrix} \quad .$$

We can now take the $P_1 = \langle [P',U_1];P_1'\rangle$ and construct

$$\mathsf{q}{\downarrow}\,\dfrac{\begin{matrix}\langle [P',U_1];P_1';P_2\rangle\\ {\scriptstyle\Delta_2}\|\\ \langle [P',U_1];[P'',U_2]\rangle\end{matrix}}{\begin{matrix}[\langle P';P''\rangle,\langle U_1;U_2\rangle]\\ {\scriptstyle\Delta_1}\|\\ [\langle P';P''\rangle,U]\end{matrix}} \qquad \text{and} \qquad \mathsf{q}{\downarrow}\,\dfrac{\vdots\,\pi}{\langle [R',P',U_1];[R'',P_1']\rangle}}{[\langle R';R''\rangle,\langle [P',U_1];P_1'\rangle]} \quad .$$

A similar argument holds when $T = \langle T';T''\rangle$ and we have a proof

$$\mathsf{q}{\downarrow}\,\dfrac{\vdots\,\pi}{[\langle [\langle R;T'\rangle,P'];[T'',P'']\rangle,U]}}{[\langle R;T';T''\rangle,\langle P';P''\rangle,U]} \quad .$$

(ii) $P = [\langle P'; P''\rangle, U', U'']$ and

$$\mathsf{q}{\downarrow}\,\frac{\vdots\,\Pi}{\dfrac{[\langle[\langle R;T\rangle, P', U']; P''\rangle, U'']}{[\langle R;T\rangle, \langle P'; P''\rangle, U', U'']}} \quad .$$

We can apply the induction hypothesis, by Remark 5.13, and we get

$$\begin{array}{c}\langle U_1; U_2\rangle\\ \|\\ U''\end{array} \quad , \quad \begin{array}{c}\Pi_1\\ \vdots\\ [\langle R;T\rangle, P', U', U_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi_2\\ \vdots\\ [P'', U_2]\end{array} \quad .$$

Because $|\downarrow_{[\langle R;T\rangle, P', U', U_1]}| < |\downarrow_{[\langle R;T\rangle, \langle P'; P''\rangle, U', U'']}|$ (otherwise the $\mathsf{q}{\downarrow}$ instance would be trivial), we can apply the induction hypothesis on Π_1, by Remark 5.13, and get

$$\begin{array}{c}\langle P_1; P_2\rangle\\ \|\\ [P', U', U_1]\end{array} \quad , \quad \begin{array}{c}\Pi_3\\ \vdots\\ [R, P_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi_4\\ \vdots\\ [T, P_2]\end{array} \quad .$$

We can now construct

$$\begin{array}{c}\langle P_1; P_2\rangle\\ \|\\ [P', U_1, U']\\ \|\\ \mathsf{q}{\downarrow}\,\dfrac{[\langle[P', U_1]; [P'', U_2]\rangle, U']}{[\langle P'; P''\rangle, U', \langle U_1; U_2\rangle]}\\ \|\\ [\langle P'; P''\rangle, U', U'']\end{array} \quad .$$

A similar argument holds when we have a proof

$$\mathsf{q}{\downarrow}\,\frac{\vdots\,\Pi}{\dfrac{[\langle P'; [\langle R;T\rangle, P'', U']\rangle, U'']}{[\langle R;T\rangle, \langle P'; P''\rangle, U', U'']}} \quad .$$

(c) $\rho = \mathsf{lis}$: If the redex is inside R, T or P, we have analogous situations to the ones seen in Case 1.a. The only other possibility is the following: Let $P = [(P', P''), U]$ and we have the proof

$$\mathsf{lis}\,\frac{\vdots\,\Pi}{\dfrac{[([\langle R;T\rangle, P'], P''), U]}{[\langle R;T\rangle, (P', P''), U]}} \quad .$$

We can apply the induction hypothesis, by Remark 5.12, and we get

$$\begin{array}{c}[U_1,U_2]\\ {\scriptstyle\Delta_1}\|\\ U\end{array}\quad,\quad\begin{array}{c}\\ \Pi_1\|\\ [\langle R;T\rangle,P',U_1]\end{array}\quad\text{and}\quad\begin{array}{c}\\ \Pi_2\|\\ [P'',U_2]\end{array}\quad.$$

Because $|\downarrow_{[\langle R;T\rangle,P',U',U_1]}| < |\downarrow_{[\langle R;T\rangle,(P',P''),U',U'']}|$ (otherwise the lis instance would be trivial), we can apply the induction hypothesis on Π_1, by Remark 5.12, and get

$$\begin{array}{c}\langle P_1;P_2\rangle\\ \|\\ [P',U_1]\end{array}\quad,\quad\begin{array}{c}\\ \Pi_3\|\\ [R,P_1]\end{array}\quad\text{and}\quad\begin{array}{c}\\ \Pi_4\|\\ [T,P_2]\end{array}\quad.$$

We can now construct

$$\begin{array}{c}\langle P_1;P_2\rangle\\ \|\\ [P',U_1]\\ {\scriptstyle\Delta}\|\\ [(P',P''),U_1,U_2]\\ \|\\ [(P',P''),U]\end{array}\quad;$$

where Δ is the derivation delivered by Lemma 5.24 with proof Π_2.

(2) $\forall n'.(n' \prec n \wedge \mathsf{S}(n') \wedge \mathsf{C}(n')) \Rightarrow \mathsf{C}(n)$, that is, $|\downarrow_{[(R,T),P]}| = n$ and $[(R,T),P]$ has a proof. Consider the bottom rule instance in this proof:

$$\rho\,\dfrac{\begin{array}{c}\Pi\|\\ Q\end{array}}{[(R,T),P]}\quad.$$

Again I assume that ρ is non-trivial, because every proof with trivial rule instances can be rewritten as a proof where these trivial instances are removed. Let us do case analysis on the position of the redex of ρ in $[(R,T),P]$. We have the following possibilities:

(a) $\rho = \mathsf{ai}{\downarrow}$: Analogous to Case 1.a.

(b) $\rho = \mathsf{q}{\downarrow}$: If the redex is inside R, T or P, the situation is analogous to the ones in Case 1.a. The only other possibility is the following; let $P = [\langle P';P''\rangle,U',U'']$ and the given proof be

$$\mathsf{q}{\downarrow}\,\dfrac{\begin{array}{c}\Pi\|\\ [\langle[(R,T),P',U'];P''\rangle,U'']\end{array}}{[(R,T),\langle P';P''\rangle,U',U'']}\quad.$$

We can apply the induction hypothesis, by Remark 5.13, and we get

$$\begin{array}{c}\langle U_1;U_2\rangle\\ \|\\ U''\end{array}\quad,\quad\begin{array}{c}\\ \Pi_1\|\\ [(R,T),P',U',U_1]\end{array}\quad\text{and}\quad\begin{array}{c}\\ \Pi_2\|\\ [P'',U_2]\end{array}\quad.$$

Because $|\downarrow_{[(R,T),P',U',U_1]}| < |\downarrow_{[(R,T),\langle P';P''\rangle,U',U'']}|$ (otherwise the $\mathsf{q}\downarrow$ instance would be trivial), we can apply the induction hypothesis on Π_1, by Remark 5.13, and get

$$\begin{array}{c}[P_1,P_2]\\ \|\\ [P',U',U_1]\end{array} \quad , \quad \begin{array}{c}\Pi_3\\ \vdots\\ [R,P_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi_4\\ \vdots\\ [T,P_2]\end{array} \quad .$$

We can now construct

$$\begin{array}{c}[P_1,P_2]\\ \|\\ [P',U',U_1]\\ \|\\ \mathsf{q}\downarrow\dfrac{[\langle[P',U_1];[P'',U_2]\rangle,U']}{[\langle P';P''\rangle,U',\langle U_1;U_2\rangle]}\\ \|\\ [\langle P';P''\rangle,U',U'']\end{array} \quad .$$

A similar argument holds when we have a proof

$$\mathsf{q}\downarrow\dfrac{\begin{array}{c}\vdots\\ [\langle P';[(R,T),P'',U']\rangle,U'']\end{array}}{[(R,T),\langle P';P''\rangle,U',U'']} \quad .$$

(c) $\rho = \mathsf{lis}$: If the redex is inside R,T or P, we have analogous situations to the ones seen in Case 1.a. The following cases exhaust the possibilities:

(i) $R=(R',R''),T=(T',T''),P=[P',P'']$ and

$$\mathsf{lis}\dfrac{\begin{array}{c}\vdots\\ [([(R',T'),P'],R'',T''),P'']\end{array}}{[(R',R'',T',T''),P',P'']} \quad :$$

We can apply the induction hypothesis, by Remark 5.12, and we get

$$\begin{array}{c}[P'_1,P'_2]\\ \Delta_1\|\\ P''\end{array} \quad , \quad \begin{array}{c}\Pi\\ \vdots\\ [(R',T'),P',P'_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi'\\ \vdots\\ [(R'',T''),P'_2]\end{array} \quad .$$

Because $|\downarrow_{[(R',T'),P',P'_1]}| < |\downarrow_{[(R',R'',T',T''),P',P'']}| > |\downarrow_{[(R'',T''),P'_2]}|$ we can apply the induction hypothesis both on Π and Π', by Remark 5.12, and get

$$\begin{array}{c}[P''_1,P''_2]\\ \Delta_2\|\\ [P',P'_1]\end{array} \quad , \quad \begin{array}{c}\Pi_1\\ \vdots\\ [R',P''_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi_2\\ \vdots\\ [T',P''_2]\end{array} \quad .$$

$$\begin{array}{c}[P_1''', P_2'''] \\ {\scriptstyle \Delta_3}\| \\ P_2'\end{array} \quad , \quad \begin{array}{c}\\ {\scriptstyle \Pi_3}\| \\ [R'', P_1''']\end{array} \quad \text{and} \quad \begin{array}{c}\\ {\scriptstyle \Pi_4}\| \\ [T'', P_2''']\end{array} \quad .$$

We can now take $P_1 = [P_1'', P_1''']$, and $P_2 = [P_2'', P_2''']$ and construct

$$\begin{array}{c}[P_1'', P_2'', P_1''', P_2'''] \\ {\scriptstyle \Delta_3}\| \\ [P_1'', P_2'', P_2'] \\ {\scriptstyle \Delta_2}\| \\ [P', P_1', P_2'] \\ {\scriptstyle \Delta_1}\| \\ [P', P'']\end{array} \quad , \quad \begin{array}{c}\| \\ [R'', P_1'''] \\ {\scriptstyle \Delta_4}\| \\ [(R', R''), P_1'', P_1''']\end{array} \quad , \text{ and } \quad \begin{array}{c}\| \\ [T'', P_2'''] \\ {\scriptstyle \Delta_5}\| \\ [(T', T''), P_2'', P_2''']\end{array} \quad ,$$

where Δ_4 is the derivation delivered by Lemma 5.24 with proof Π_3, and Δ_5 is the derivation delivered by Lemma 5.24 with proof Π_4.

(ii) $P = [(P', P''), U]$ and

$$\mathsf{lis}\,\frac{\begin{array}{c}\| \\ [([(R,T), P'], P''), U]\end{array}}{[(R,T), (P', P''), U]} \quad :$$

We can apply the induction hypothesis, by Remark 5.12, and we get

$$\begin{array}{c}[U_1, U_2] \\ \| \\ U\end{array} \quad , \quad \begin{array}{c}\\ {\scriptstyle \Pi_1}\| \\ [(R,T), P', U_1]\end{array} \quad \text{and} \quad \begin{array}{c}\\ {\scriptstyle \Pi_2}\| \\ [P'', U_2]\end{array} \quad .$$

Because $|\downarrow_{[(R,T),P',U_1]}| < |\downarrow_{[(R,T),(P',P''),U]}|$ (otherwise the lis instances would be trivial), we can apply the induction hypothesis on Π_1, by Remark 5.12, and get

$$\begin{array}{c}[P_1, P_2] \\ \| \\ [P', U_1]\end{array} \quad , \quad \begin{array}{c}\\ {\scriptstyle \Pi_3}\| \\ [R, P_1]\end{array} \quad \text{and} \quad \begin{array}{c}\\ {\scriptstyle \Pi_4}\| \\ [T, P_2]\end{array} \quad .$$

We can now construct

$$\begin{array}{c}[P_1, P_2] \\ \| \\ [P', U_1] \\ {\scriptstyle \Delta}\| \\ [(P', P''), U_1, U_2] \\ \| \\ [(P', P''), U]\end{array} \quad .$$

where Δ is the derivation delivered by Lemma 5.24 with proof Π_2.

□

As we have seen in Proposition 2.30, system BV is a conservative extension of system FBV. This observation allows to carry the ideas above to system FBV.

Definition 5.26. *System* FBV *with interaction switch, or system* FBVs*, is the system resulting from replacing the rule* s *in* FBV *with the rule* is*.*

Definition 5.27. *System* FBV *with lazy interaction switch, or system* FBVi*, is the system resulting from replacing the rule* s *in* FBV *with the rule* lis*.*

Corollary 5.28. *(Shallow Splitting for* FBVi*) For all structures R, T and P, if $[(R,T),P]$ is provable in* FBVi *then there exists P_1, P_2 and* $\Delta \vdots \dfrac{[P_1,P_2]}{P}\,\mathsf{FBVi}$ *such that $[R,P_1]$ and $[T,P_2]$ are provable in* FBVi*.*

Proof. Follow the steps of the Theorem 5.25, leaving out the cases that involve the seq operator. □

Because inference rules can be applied at any depth inside a structure, we need the following theorem for accessing the deeper structures. This theorem is a specialization of the context reduction theorem for BV in [**Gug07**]. In the following, this theorem will be useful to reduce the context of a substructure of a provable structure to the same level as the substructure without losing provability while going up in a derivation.

Theorem 5.29. (Context reduction for BVsl) *For all structures R and for all contexts $S\{\ \}$ such that $S\{R\}$ is provable in* BVsl*, there exists a structure U such that for all structures X there exist derivations:*

$$\dfrac{[X,U]}{S\{X\}}\,\mathsf{BVsl} \qquad \textit{and} \qquad \dfrac{\top}{[R,U]}\,\mathsf{BVsl} \quad .$$

Proof. We prove by induction on the size of $S\{\circ\}$. For the base, where $S\{\circ\} = \circ$, we get $U = \circ$. There are three inductive cases:

(1) $S\{\ \} = \langle S'\{\ \}; P\rangle$ where $P \neq \circ$: By Proposition 5.20 there are proofs in BVsl of $S'\{R\}$ and of P. By applying the induction hypothesis, we can find U and construct, for all X, the derivation

$$\dfrac{\dfrac{[X,U]}{S'\{X\}}\,\mathsf{BVsl}}{\langle S'\{X\}; P\rangle}\,\mathsf{BVsl}$$

such that $[R,U]$ is provable in BVsl. We can apply the same argument for the case where $S\{\ \} = \langle P; S'\{\ \}\rangle$ and $P \neq \circ$.

(2) $S\{\ \} = [S'\{\ \}, P]$ where $P \neq \circ$ such that $S'\{\ \}$ is not a proper par: If $S'\{\circ\} = \circ$ then the theorem is proved. Otherwise there are the following two possibilities:
   (a) $S'\{\ \} = (S''\{\ \}, P')$ where $P \neq \circ$: By Theorem 5.25 there exist structures P_1 and P_2, the derivation

$$\begin{array}{c}[P_1, P_2]\\ \llbracket \mathsf{BVsl}\\ P\end{array}\ ,\quad \text{and the proofs}\quad \begin{array}{c}\Pi_1\\ \Vert\!\!\Vert_{\mathsf{BVsl}}\\ [S''\{R\}, P_1]\end{array}\quad \text{and}\quad \begin{array}{c}\Pi_2\\ \Vert\!\!\Vert_{\mathsf{BVsl}}\\ [P', P_2]\end{array}\ .$$

   By applying the induction hypothesis to Π_1 we get a derivation Δ_1. Then we can construct

$$\begin{array}{c}[X, U]\\ \Delta_1\,\llbracket \mathsf{BVsl}\\ [S''\{X\}, P_1]\\ \Delta\,\llbracket \mathsf{BVsl}\\ [(S'\{X\}, P'), P_1, P_2]\\ \llbracket \mathsf{BVsl}\\ [(S''\{X\}, P'), P]\end{array}\qquad \text{and}\qquad \begin{array}{c}\Vert\!\!\Vert_{\mathsf{BVs}}\\ [R, U]\end{array}\ .$$

   where Δ is the derivation delivered by Lemma 5.24 with proof Π_2.
   (b) $S'\{\ \} = \langle S''\{\ \}; P'\rangle$ where $P' \neq \circ$: By Theorem 5.25 there exist the structures P_1 and P_2, the derivation

$$\begin{array}{c}\langle P_1; P_2\rangle\\ \llbracket \mathsf{BVsl}\\ P\end{array}\ ,\quad \text{and the proofs}\quad \begin{array}{c}\Pi_1\\ \Vert\!\!\Vert_{\mathsf{BVsl}}\\ [S''\{R\}, P_1]\end{array}\quad \text{and}\quad \begin{array}{c}\Pi_2\\ \Vert\!\!\Vert_{\mathsf{BVsl}}\\ [P', P_2]\end{array}\ .$$

   By applying the induction hypothesis to Π_1 we get a derivation Δ_1. Then we can construct

$$\begin{array}{c}[X, U]\\ \Delta_1\,\llbracket \mathsf{BVsl}\\ [S''\{X\}, P_1]\\ \Pi_2\,\llbracket \mathsf{BVsl}\\ \mathsf{q}{\downarrow}\,\dfrac{\langle [S''\{X\}, P_1]; [P', P_2]\rangle}{[\langle S''\{X\}; P'\rangle, \langle P_1; P_2\rangle]}\\ \llbracket \mathsf{BVsl}\\ [\langle S''\{X\}; P'\rangle, P]\end{array}\qquad \text{and}\qquad \begin{array}{c}\Vert\!\!\Vert_{\mathsf{BVs}}\\ [R, U]\end{array}\ .$$

   We can proceed analogously when $S'\{\ \} = \langle P'; S''\{\ \}\rangle$ where $P' \neq \circ$.
(3) $S'\{\ \} = (S'\{\ \}, P)$ where $P \neq \circ$: Analogous to Case 1.

□

Corollary 5.30. (Context reduction for FBVi) *For all structures R and for all contexts $S\{\ \}$ such that $S\{R\}$ is provable in* FBVi*, there exists a structure U such that for all structures X there exist derivations:*

$$\begin{array}{c}[X, U]\\ \llbracket \mathsf{FBVi}\\ S\{X\}\end{array}\qquad \textit{and}\qquad \begin{array}{c}\Vert\!\!\Vert_{\mathsf{FBVi}}\\ [R, U]\end{array}\ .$$

PROOF. Analogous to the proof of Theorem 5.29 by using Corollary 5.28 instead of Theorem 5.25. □

COROLLARY 5.31. (Splitting for system BVsl) *For all structures R, T and for all contexts $S\{\ \}$:*

(1) *if $S\langle R;T\rangle$ is provable in* BVsl *then there exist structures S_1 and S_2 such that, for all structures X, there exists a derivation*

$$\begin{array}{c}[X,\langle S_1;S_2\rangle]\\ \Delta\,\|_{\mathsf{BVsl}}\\ S\{\,X\,\}\end{array}\quad;$$

(2) *if $S(R,T)$ is provable in* BVsl *then there exist structures S_1 and S_2 such that, for all structures X, there exists a derivation*

$$\begin{array}{c}[X,S_1,S_2]\\ \Delta\,\|_{\mathsf{BVsl}}\\ S\{\,X\,\}\end{array}\quad;$$

and, in both cases, there are proofs $\begin{array}{c}\Pi_1\\ \|_{\mathsf{BVsl}}\\ [R,S_1]\end{array}$ *and* $\begin{array}{c}\Pi_2\\ \|_{\mathsf{BVsl}}\\ [T,S_2]\end{array}$.

PROOF. The proof for (2) being analogous, the proof of (1) is as follows: Given that $S\langle R;T\rangle$ is provable in BVsl, apply Theorem 5.29 to obtain the derivation Δ. Replace X in Δ with $\langle R;T\rangle$. From Theorem 5.29, it follows that $[\langle R;T\rangle,\langle S_1;S_2\rangle]$ has a proof Π. Apply Theorem 5.25 to Π to obtain the proofs Π_1 and Π_2. □

We can now state the following results:

THEOREM 5.32. *Systems* BV *and* BVsl *are equivalent.*

PROOF. Observe that every proof in BVsl is also a proof in BV. For the other direction, single out the upper-most instance of the switch rule in the BV proof which is not an instance of the lazy interaction switch rule:

$$\mathsf{s}\,\frac{\begin{array}{c}\\ \|_{\mathsf{BVsl}}\\ S([R,U],T)\end{array}}{S[(R,T),U]}$$

From Theorem 5.29, there exists a structure V and a derivation

$$\begin{array}{c}[\{\ \},V]\\ \|_{\mathsf{BVsl}}\\ S\{\ \}\end{array}\quad\text{such that}\quad\begin{array}{c}\\ \|_{\mathsf{BVsl}}\\ [([R,U],T),V]\end{array}.$$

It follows from Theorem 5.25 that there are structures K_1 and K_2, a derivation

$$\begin{array}{c}[K_1,K_2]\\ \|_{\mathsf{BVsl}}\\ V\end{array},\quad\text{and proofs}\quad\begin{array}{c}\\ \|_{\mathsf{BVsl}}\\ [R,U,K_1]\end{array},\quad\text{and}\quad\begin{array}{c}\Pi\\ \|_{\mathsf{BVsl}}\\ [K_2,T]\end{array}.$$

We can then construct the following proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{[R,U,K_1]}\,\mathsf{BVsl}}{[(R,T),U,K_1,K_2]}\,\Delta\,[\,\mathsf{BVsl}}{[(R,T),U,V]}\,[\,\mathsf{BVsl}}{S[(R,T),U]}\,[\,\mathsf{BVsl}$$

where Δ is the derivation delivered by Lemma 5.24 with proof Π. Repeat the above procedure inductively until all the instances of the switch rule that are not instances of lazy interaction switch rule are removed. □

COROLLARY 5.33. *Systems* BV *and* BVs *are equivalent.*

PROOF. Observe that every proof in BVsl is a proof in BVs, and every proof in BVs is a proof in BV. □

COROLLARY 5.34. *Systems* FBV*,* FBVs *and* FBVi *are equivalent.*

PROOF. The proof of equivalence of FBV and FBVi is analogous to the proof of Theorem 5.32 by using Corollary 5.28 and Corollary 5.30. Observe that every proof in FBVi is a proof in FBVs, and every proof in FBVs is a proof in FBV. □

EXAMPLE 5.35. *Consider the provable* FBV *structure* $[(\bar{a},\bar{b}),a,b]$ *of Example 5.1. The only proofs of this structure in system* FBV*, and also in* FBVi*, are the following:*

$$\mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[b,\bar{b}]}}{[([a,\bar{a}],\bar{b}),b]}}{[(\bar{a},\bar{b}),a,b]}\qquad \mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[a,\bar{a}]}}{([a,\bar{a}],[b,\bar{b}])}}{[([a,\bar{a}],\bar{b}),b]}}{[(\bar{a},\bar{b}),a,b]}\qquad \mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[b,\bar{b}]}}{([a,\bar{a}],[b,\bar{b}])}}{[([a,\bar{a}],\bar{b}),b]}}{[(\bar{a},\bar{b}),a,b]}$$

$$\mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[a,\bar{a}]}}{[a,([a,\bar{a}],\bar{b})]}}{[(\bar{a},\bar{b}),a,b]}\qquad \mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[b,\bar{b}]}}{([a,\bar{a}],[b,\bar{b}])}}{[a,([a,\bar{a}],\bar{b})]}}{[(\bar{a},\bar{b}),a,b]}\qquad \mathsf{ai}{\downarrow}\dfrac{\mathsf{lis}\dfrac{\mathsf{lis}\dfrac{\mathsf{ai}{\downarrow}\dfrac{\circ}{[a,\bar{a}]}}{([a,\bar{a}],[b,\bar{b}])}}{[a,([a,\bar{a}],\bar{b})]}}{[(\bar{a},\bar{b}),a,b]}$$

In system FBV*, in the proof search space of* $[(\bar{a},\bar{b}),a,b]$*, there are 358 derivations including these 6 proofs and no other proofs. However, in system* FBVi *these 6 proofs are the only possible derivations.*

Using the above results, I will now prove the cut elimination result for system BVsl, similar to the proof of cut elimination for system BV in [**Gug07**]. The following proposition will be necessary.

PROPOSITION 5.36. *For every context $S\{\ \}$ and structures R and T, there exists a derivation*

$$\begin{array}{c} S[R,T] \\ \Vert\mathsf{BV} \\ [S\{R\},T] \end{array} .$$

PROOF. Proof with structural induction on $S\{\ \}$: Base case, where $S\{\ \}$ is the empty context, is trivial. There are three inductive cases:

(1) If $S\{\ \} = \langle S'\{\ \}; P\rangle$ then take the derivation

$$\mathsf{q}{\downarrow}\dfrac{\begin{array}{c}\langle S'[R,T];P\rangle \\ \Delta\Vert\mathsf{BV} \\ \langle [S'\{R\},T];P\rangle\end{array}}{[\langle S'\{R\};P\rangle,T]}$$

where the derivation Δ is delivered by the induction hypothesis.

(2) If $S\{\ \} = (S'\{\ \}, P)$ then take the derivation

$$\mathsf{s}\dfrac{\begin{array}{c}(S'[R,T],P) \\ \Delta\Vert\mathsf{BV} \\ ([S'\{R\},T],P)\end{array}}{[(S'\{R\},P),T]}$$

where the derivation Δ is delivered by the induction hypothesis.

(3) If $S\{\ \} = [S'\{\ \}, P]$ then take the derivation

$$\approx\dfrac{\begin{array}{c}[S'[R,T],P] \\ \Delta\Vert\mathsf{BV} \\ [[S'\{R\},T],P]\end{array}}{[[S'\{R\},P],T]}$$

where the derivation Δ is delivered by the induction hypothesis.

□

THEOREM 5.37. (Cut elimination for system BVsl) *The cut rule (*$\mathsf{ai}{\uparrow}$*) is admissible for system* BVsl.

PROOF. Consider the proof

$$\mathsf{ai}{\uparrow}\dfrac{\begin{array}{c}\Pi\Vert\mathsf{BVsl} \\ S(a,\bar{a})\end{array}}{S\{\circ\}} .$$

By applying Corollary 5.31 to proof Π we get the following derivations:

$$\begin{array}{c}[S_1,S_2] \\ \Delta\Vert\mathsf{BVsl} \\ S\{\circ\}\end{array} , \quad \begin{array}{c}\Pi_1\Vert\mathsf{BVsl} \\ [a,S_1]\end{array} \quad \text{and} \quad \begin{array}{c}\Pi_2\Vert\mathsf{BVsl} \\ [\bar{a},S_2]\end{array} .$$

It follows that in proof Π_1 there must be a context $S'_1\{\ \ \}$ such that $S_1 = S'_1\{\,\bar{a}\,\}$ and

$$\Pi_1 = \begin{array}{c} \top\!\!\top_{\mathsf{BVsl}} \\ \mathsf{ai}{\downarrow}\dfrac{S''\{\circ\}}{S''[a,\bar{a}]} \\ \Delta\,\|_{\mathsf{BVsl}} \\ [a, S'_1\{\bar{a}\}] \end{array}$$

for some $S''\{\ \ \}$, in which we single out the instance of the rule $\mathsf{ai}{\downarrow}$ where the occurrence of a interacts with the occurrence of $\bar{a}$ from $S'_1\{\,\bar{a}\,\}$. By replacing in Π_1 every occurrence of a and $\bar{a}$ with $\circ$, we can obtain a proof in BVsl of $S'_1\{\,\circ\,\}$. Analogously, we can transform Π_2 into a proof in BVsl of $S'_2\{\circ\}$ such that $S_2 = S'_2\{a\}$. Because every proof in BVsl is also a proof in BV, we can then construct the following proof

$$\begin{array}{c} \top\!\!\top_{\mathsf{BVsl}} \\ S'_1\{\,\circ\,\} \\ \|_{\mathsf{BVsl}} \\ \mathsf{ai}{\downarrow}\dfrac{S'_1\{S'_2\{\,\circ\,\}\}}{S'_1\{S'_2[a,\bar{a}]\}} \\ \Delta'\,\|_{\mathsf{BV}} \\ [S'_1\{\,\bar{a}\,\}, S'_2\{\,a\,\}] \\ \Delta\,\|_{\mathsf{BVsl}} \\ S\{\,\circ\,\} \end{array}$$

where Proposition 5.36 is used twice to construct the derivation Δ'. By applying Theorem 5.32, replace the proof of $[S'_1\{\,\bar{a}\,\}, S'_2\{\,a\,\}]$ in BV above with a proof in BVsl. Repeat this argument inductively, starting from the top-most instance of the rule $\mathsf{ai}{\uparrow}$, for any proof in $\mathsf{BVsl} \cup \{\,\mathsf{ai}{\uparrow}\,\}$ and eliminate all the instances of the rule $\mathsf{ai}{\uparrow}$ one after another. □

5.3. The Seq Rule

At a first glance, the rules switch and seq appear to be different in nature due to the different logical operators they work on. However, at a closer inspection of these rules, one can observe that both of these rules manage the context of the structures they are applied at in a similar way. While the switch rule reduces the interaction in the structures involving a copar structure in a bottom-up application, the seq rule does the same with the structures involving seq structures. In this section, exploiting this observation, I will carry the ideas from the previous section to the seq rule at the level of conjecture, state some facts about system BV, and discuss the difficulties in attempts for proving this conjecture.

Definition 5.38. *The rules*

$$\mathsf{lq}_3{\downarrow}\,\frac{S\langle[R,W];T\rangle}{S[\langle R;T\rangle,W]} \qquad \textit{and} \qquad \mathsf{lq}_4{\downarrow}\,\frac{S\langle R;[T,W]\rangle}{S[\langle R;T\rangle,W]}\ ,$$

where W is not a proper par structure, are called lazy seq 3 *($\mathsf{lq}_3\downarrow$) and* lazy seq 4 *($\mathsf{lq}_4\downarrow$), respectively.*

PROPOSITION 5.39. *Let $\mathscr{S} \in \{\mathsf{BV}, \mathsf{BVs}, \mathsf{BVsl}\}$. The system resulting from replacing the rule $\mathsf{q}\downarrow$ in $\mathscr{S}$ with $\{\mathsf{q}_1\downarrow, \mathsf{q}_2\downarrow, \mathsf{lq}_3\downarrow, \mathsf{lq}_4\downarrow\}$ and system BV are equivalent.*

PROOF. The rules $\mathsf{q}_3\downarrow$ and $\mathsf{q}_4\downarrow$ are derivable for the rules $\mathsf{lq}_3\downarrow$ and $\mathsf{lq}_4\downarrow$:

$$\mathsf{lq}_3\downarrow \frac{S\langle [R,V,T];U\rangle}{S[\langle [V,T];U\rangle, R]} \qquad \mathsf{lq}_4\downarrow \frac{S\langle T;[R,V,U]\rangle}{S[\langle T;[V,U]\rangle, R]}$$
$$\mathsf{lq}_3\downarrow \frac{S[\langle T;U\rangle, R, V]}{S[\langle T;U\rangle, R, V]} \qquad \mathsf{lq}_4\downarrow \frac{S[\langle T;U\rangle, R, V]}{S[\langle T;U\rangle, R, V]}$$

Result follows immediately from Corollary 4.29. □

DEFINITION 5.40. *Let W denote structures that are not proper par structures. The following rules are called* interaction seq rule 1*,* lazy interaction seq rule 3*, and* lazy interaction seq rule 4*, respectively:*

$$\mathsf{iq}_1\downarrow \frac{S\langle [R,U];[T,V]\rangle}{S[\langle R;T\rangle, \langle U;V\rangle]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,U \neq \emptyset \quad \textit{and} \quad \mathsf{at}\,\overline{T} \cap \mathsf{at}\,V \neq \emptyset$$

$$\mathsf{liq}_3\downarrow \frac{S\langle [R,W];T\rangle}{S[\langle R;T\rangle, W]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,W \neq \emptyset$$

$$\mathsf{liq}_4\downarrow \frac{S\langle T;[R,W]\rangle}{S[\langle T;R\rangle, W]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,W \neq \emptyset$$

DEFINITION 5.41. *The system resulting from replacing the seq rule in system* BVsl *with the rules $\mathsf{iq}_1\downarrow$, $\mathsf{q}_2\downarrow$, $\mathsf{liq}_3\downarrow$, and $\mathsf{liq}_4\downarrow$ is called* interaction system BV*, or* BVi*.*

DEFINITION 5.42. *Let W denote structures that are not proper par structures. The following rules are called* non-interaction seq rule 1*,* non-interaction seq rule 3*, and* non-interaction seq rule 4*, respectively:*

$$\mathsf{niq}_1\downarrow \frac{S\langle [R,U];[T,V]\rangle}{S[\langle R;T\rangle, \langle U;V\rangle]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,U = \emptyset \ \textit{or}\ \mathsf{at}\,\overline{T} \cap \mathsf{at}\,V = \emptyset$$

$$\mathsf{niq}_3\downarrow \frac{S\langle [R,W];T\rangle}{S[\langle R;T\rangle, W]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,W = \emptyset$$

$$\mathsf{niq}_4\downarrow \frac{S\langle T;[R,W]\rangle}{S[\langle T;R\rangle, W]} \quad \textit{where} \quad \mathsf{at}\,\overline{R} \cap \mathsf{at}\,W = \emptyset$$

REMARK 5.43. *Every instance of the rule $\mathsf{q}\downarrow$ is an instance of one of the rules $\mathsf{iq}_1\downarrow$, $\mathsf{niq}_1\downarrow$, $\mathsf{q}_2\downarrow$, $\mathsf{liq}_3\downarrow$, $\mathsf{niq}_3\downarrow$, $\mathsf{liq}_4\downarrow$, or $\mathsf{niq}_4\downarrow$. We have seen in Proposition 5.39 that the systems $\{\mathsf{q}\downarrow\}$ and $\{\mathsf{q}_1\downarrow, \mathsf{q}_2\downarrow, \mathsf{liq}_3\downarrow, \mathsf{liq}_4\downarrow\}$ are equivalent. Every instance of the rule $\mathsf{q}_1\downarrow$ is either an instance of the rule $\mathsf{iq}_1\downarrow$ or $\mathsf{niq}_1\downarrow$, every instance of the rule $\mathsf{q}_3\downarrow$ is either an instance of the rule $\mathsf{liq}_3\downarrow$ or $\mathsf{niq}_3\downarrow$, and every instance of the rule $\mathsf{q}_4\downarrow$ is either an instance of the rule $\mathsf{liq}_4\downarrow$ or $\mathsf{niq}_4\downarrow$.*

When we carry the ideas above to system BVi, we observe that using the splitting technique will not be possible in the context of system BVi, as demonstrated by the example below:

EXAMPLE 5.44. *Consider the structure*

$$[\langle [a,b,c];[d,e]\rangle, \bar{a}, \langle \bar{b};\bar{d}\rangle, \langle \bar{c};\bar{e}\rangle]$$

which is provable in BVsl*. By applying Theorem 5.25, we can obtain the derivation*

$$\Delta = \begin{array}{l}\mathsf{q}_3\downarrow \dfrac{\langle [\bar{a},\bar{b},\bar{c}];[\bar{d},\bar{e}]\rangle}{[\bar{a},\langle [\bar{b},\bar{c}];[\bar{d},\bar{e}]\rangle]} \\ \mathsf{q}_1\downarrow \dfrac{}{[\bar{a},\langle \bar{b};\bar{d}\rangle,\langle \bar{c};\bar{e}\rangle]}\end{array} \quad \textit{such that} \quad \begin{array}{c}\prod\!\!\!\!\prod^{\mathsf{BVsl}} \\ [\bar{a},\bar{b},\bar{c},a,b,c]\end{array} \quad \textit{and} \quad \begin{array}{c}\prod\!\!\!\!\prod^{\mathsf{BVsl}} \\ [\bar{d},\bar{e},d,e]\end{array} .$$

However, the derivation Δ *is impossible in system* BVi *because the instances of the rules* $\mathsf{q}_1\downarrow$ *and* $\mathsf{q}_3\downarrow$ *in* Δ *are instances of the rules* $\mathsf{niq}_1\downarrow$ *and* $\mathsf{niq}_3\downarrow$*, respectively.*

In the calculus of structures, often inference rules can be permuted over each other, for example instance of one rule is inside the context of the other. In his Ph.D. thesis, Straßburger gives a characterization of such permutations [**Str03a**]. Let me now consider these permutations in the context of system BVi:

DEFINITION 5.45. *A rule* ρ *permutes over a rule* β *(or* β *permutes under* ρ*) if for every derivation* $\begin{array}{l}\beta\dfrac{Q}{U}\\ \rho\dfrac{}{P}\end{array}$ *there is a derivation* $\begin{array}{l}\rho\dfrac{Q}{V}\\ \beta\dfrac{}{P}\end{array}$.

DEFINITION 5.46. *A rule* ρ *permutes well over a rule* β *if for every derivation* $\begin{array}{l}\beta\dfrac{Q}{U}\\ \rho\dfrac{}{P}\end{array}$ *there is a derivation* $\begin{array}{l}\rho\dfrac{Q}{V}\\ \beta\dfrac{}{P}\end{array}$ *or* $\rho\dfrac{Q}{P}$.

DEFINITION 5.47. *In an instance of an inference rule a substructure that occurs exactly once in the redex as well as in the contractum of a rule without changing is called* passive*, and all the substructures of the redexes and the contracta, that are not passive, (i.e. that change, disappear or are duplicated) are called* active*.*

EXAMPLE 5.48. *Consider the following instance of the rule* s*:*

$$\mathsf{s}\,\frac{([a,b,d],c)}{[([a,b],c),d]}$$

In this instance, the structures a*,* b*,* c*,* d*, and* $[a,b]$ *are passive, whereas the structures* $([a,b,d],c)$*,* $[([a,b],c),d]$*,* $[a,b,d]$*,* $[b,d]$*,* $[a,d]$*, and* $([a,b],c)$ *are active.*

REMARK 5.49. *Every rule* ρ *permutes over every rule* β *if both of the following conditions hold:*

(a) the redex of β *is not inside an active structure of the contractum of* ρ*;*
(b) the contractum of ρ *is not inside an active structure of the redex of* β*.*

The reason for this can be observed, as it is demonstrated, in the following situation:

$$\begin{array}{l}\beta\dfrac{Q}{S\{U\}}\\ \rho\dfrac{}{S\{R\}}\end{array}$$

where the redex R and the contractum U of ρ are known and we have to make a case analysis for the proposition of the redex of β inside the structure $S\{U\}$. There are the following possibilities:

(1) The redex of β is inside the context $S\{\ \}$ of ρ. Let $Q = S'\{U\}$. Then we permute as follows:

$$\begin{array}{c}\beta\dfrac{S'\{U\}}{S\{U\}}\\ \rho\dfrac{S\{U\}}{S\{R\}}\end{array} \quad\rightsquigarrow\quad \begin{array}{c}\rho\dfrac{S'\{U\}}{S'\{R\}}\\ \beta\dfrac{S'\{R\}}{S\{R\}}\end{array}$$

(2) The contractum U of ρ is inside a passive structure of the redex of β. Then we permute as in case (1).

(3) The redex of β is inside a passive structure of the contractum U of ρ. Let $R = R'\{T\}$, $U = U'\{T\}$, and $Q = S\{U'\{T'\}\}$. Then we permute as follows:

$$\begin{array}{c}\beta\dfrac{S\{U'\{T'\}\}}{S\{U'\{T\}\}}\\ \rho\dfrac{S\{U'\{T\}\}}{S\{R'\{T\}\}}\end{array} \quad\rightsquigarrow\quad \begin{array}{c}\rho\dfrac{S\{U'\{T'\}\}}{S\{R'\{T'\}\}}\\ \beta\dfrac{S\{R'\{T'\}\}}{S\{R'\{T\}\}}\end{array}$$

PROPOSITION 5.50. *Any rule* $\rho \in \{\mathsf{niq}_1{\downarrow}, \mathsf{niq}_3{\downarrow}, \mathsf{niq}_4{\downarrow}, \mathsf{q}_2{\downarrow}\}$ *permutes well over any* $\beta \in \{\mathsf{lis}, \mathsf{ai}{\downarrow}, \mathsf{liq}_3{\downarrow}, \mathsf{liq}_4{\downarrow}\}$.

PROOF. It suffices to check the cases excluded by the conditions of Remark 5.49. I will prove the result for $\rho = \mathsf{niq}_1{\downarrow}$ and $\rho = \mathsf{q}_2{\downarrow}$. The cases for $\rho = \mathsf{niq}_3{\downarrow}$ and $\rho = \mathsf{niq}_4{\downarrow}$ are similar to the case for $\rho = \mathsf{niq}_1{\downarrow}$.

(1) $\rho = \mathsf{niq}_1{\downarrow}$

(a) The redex of β is inside an active structure of the contractum of $\mathsf{niq}_1{\downarrow}$. Let $\beta \in \{\mathsf{lis}, \mathsf{ai}{\downarrow}, \mathsf{liq}_3{\downarrow}, \mathsf{liq}_4{\downarrow}\}$. Then such a derivation must be of the form

$$\begin{array}{c}\beta\dfrac{S\langle\, Q\,; [T,V]\rangle}{S\langle [R,U]; [T,V]\rangle}\\ \mathsf{niq}_1{\downarrow}\dfrac{}{S[\langle R;T\rangle, \langle U;V\rangle]}\end{array}$$

where the redex and the contractum of β are marked. However, this contradicts with $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,U = \emptyset$, thus this case is impossible.

(b) The contractum of $\mathsf{niq}_1{\downarrow}$ is inside an active structure of the redex of β: For $\beta \in \{\mathsf{lis}, \mathsf{ai}{\downarrow}\}$ this is impossible because the contractum of $\mathsf{niq}_1{\downarrow}$ is a proper seq structure, whereas the redex of β does not contain any seq structures. For $\mathsf{liq}_3{\downarrow}$, we have the following situation, the case for $\mathsf{liq}_4{\downarrow}$ is similar:

$$\begin{array}{c}\mathsf{liq}_3{\downarrow}\dfrac{S\langle [R,U,P]; [T,V]\rangle}{S[P, \langle [R,U]; [T,V]\rangle]}\\ \mathsf{niq}_1{\downarrow}\dfrac{}{S[P, \langle R;T\rangle, \langle U;V\rangle]}\end{array}$$

It must be that $\mathsf{at}\,\overline{P} \cap \mathsf{at}\,[R,U] \neq \emptyset$. If $\mathsf{at}\,\overline{P} \cap \mathsf{at}\,R \neq \emptyset$, then we permute as follows:

$$\begin{array}{c}\mathsf{niq}_1{\downarrow}\dfrac{S\langle [R,U,P]; [T,V]\rangle}{S[\langle [P,R]; T\rangle, \langle U;V\rangle]}\\ \mathsf{liq}_3{\downarrow}\dfrac{}{S[P, \langle R;T\rangle, \langle U;V\rangle]}\end{array}$$

Otherwise we permute as follows:

$$\begin{array}{l}\mathsf{niq}_1\downarrow\\ \mathsf{liq}_3\downarrow\end{array}\dfrac{S\langle[R,U,P];[T,V]\rangle}{S[\langle R;T\rangle,\langle[P,U];V\rangle]}\qquad\qquad\dfrac{S\langle[R,U,P];[T,V]\rangle}{S[P,\langle R;T\rangle,\langle U;V\rangle]}$$

(2) $\rho=\mathsf{q}_2\downarrow$

(a) The redex of β is inside an active structure of the contractum of $\mathsf{q}_2\downarrow$: For $\beta\in\{\mathsf{lis},\mathsf{ai}\downarrow,\mathsf{liq}_3\downarrow,\mathsf{liq}_4\downarrow\}$ this is impossible because the contractum of $\mathsf{q}_2\downarrow$ is a proper seq structure which cannot match the redex of β.

(b) The contractum of $\mathsf{q}_2\downarrow$ is inside an active structure of the redex of β: For $\beta\in\{\mathsf{lis},\mathsf{ai}\downarrow\}$ because β does not involve any seq structures, this is impossible. For $\beta\in\{\mathsf{liq}_3\downarrow,\mathsf{liq}_4\downarrow\}$ we can have the situation where the instance of the rule $\mathsf{liq}_3\downarrow$ ($\mathsf{liq}_4\downarrow$) can be equivalently removed as follows:

$$\begin{array}{l}\mathsf{liq}_3\downarrow\\ \mathsf{q}_2\downarrow\end{array}\dfrac{S\langle[R,P];T\rangle}{S[\langle R;T\rangle,P]}\qquad\rightsquigarrow\qquad\mathsf{q}_2\downarrow\dfrac{S\langle[R,P];T\rangle}{S[R,T,P]}$$

□

In general, the rules $\mathsf{niq}_1\downarrow,\mathsf{niq}_3\downarrow$, $\mathsf{niq}_4\downarrow$, and $\mathsf{q}_2\downarrow$ cannot permute over $\mathsf{iq}_1\downarrow$. For instance, consider the following derivations (the redexes are highlighted):

$$\begin{array}{l}\mathsf{iq}_1\downarrow\\ \mathsf{niq}_1\downarrow\end{array}\dfrac{\langle[a,\bar a,b];[c,d,\bar d]\rangle}{[\langle[a,b];[c,d]\rangle,\langle\bar a;\bar d\rangle]}\qquad\begin{array}{l}\mathsf{iq}_1\downarrow\\ \mathsf{niq}_3\downarrow\end{array}\dfrac{\langle[a,b,\bar b];[c,\bar c]\rangle}{[\langle[a,b];c\rangle,\langle\bar b;\bar c\rangle]}\qquad\begin{array}{l}\mathsf{iq}_1\downarrow\\ \mathsf{q}_2\downarrow\end{array}\dfrac{\langle[a,\bar a];[b,c,\langle\bar b;\bar c\rangle]\rangle}{[\langle a;[b,c]\rangle,\langle[\bar a,\bar b];\bar c\rangle]}$$

However, we can state the following result:

PROPOSITION 5.51. *Any rule $\rho\in\{\mathsf{niq}_1\downarrow,\mathsf{niq}_3\downarrow,\mathsf{niq}_4\downarrow,\mathsf{q}_2\downarrow\}$ permutes well over $\mathsf{iq}_1\downarrow$ if the contractum of ρ is not inside an active structure of the redex of $\mathsf{iq}_1\downarrow$.*

PROOF. We check the only case excluded by the conditions of Remark 5.49: The redex of $\mathsf{iq}_1\downarrow$ is inside an active structure of the contractum of ρ. I will prove the result for $\rho=\mathsf{niq}_1\downarrow$ and $\rho=\mathsf{q}_2\downarrow$. The cases for $\rho=\mathsf{niq}_3\downarrow$ and $\rho=\mathsf{niq}_4\downarrow$ are similar to the case for $\rho=\mathsf{niq}_1\downarrow$.

(1) If $\rho=\mathsf{niq}_1\downarrow$, then the redex of $\mathsf{iq}_1\downarrow$ is inside an active structure of the contractum of $\mathsf{niq}_1\downarrow$. Such a derivation must be of the form

$$\begin{array}{l}\mathsf{iq}_1\downarrow\\ \mathsf{niq}_1\downarrow\end{array}\dfrac{S\langle[R',U'];[R'',U'']\,;[T,V]\rangle}{S\langle[\langle R';R''\rangle,\langle U';U''\rangle];[T,V]\rangle}{S[\langle R';R'';T\rangle,\langle U';U'';V\rangle]}$$

where the redex and the contractum of $\mathsf{q}_1\downarrow$ are marked. It must be that $\mathsf{at}\,\overline{\langle R';R''\rangle}\cap\mathsf{at}\,\langle U';U''\rangle=\emptyset$ or $\mathsf{at}\,\overline{T}\cap\mathsf{at}\,V=\emptyset$. These conditions cannot both hold, because this contradicts with the conditions $\mathsf{at}\,\overline{R'}\cap\mathsf{at}\,U'\neq\emptyset$ and $\mathsf{at}\,\overline{R''}\cap\mathsf{at}\,U''\neq\emptyset$. Otherwise if $\mathsf{at}\,\overline{T}\cap\mathsf{at}\,V=\emptyset$, then we have the

situation in the above derivation. Then we permute as follows:

$$\begin{array}{c} \mathsf{niq}_1\downarrow \\ \mathsf{iq}_1\downarrow \end{array} \dfrac{\dfrac{S\langle [R',U'];[R'',U''];[T,V]\rangle}{S\langle [R',U'];[\langle R'';T\rangle,\langle U'';V\rangle]\rangle}}{S[\langle R';R'';T\rangle,\langle U';U'';V\rangle]}$$

(2) If $\rho = \mathsf{q}_2\downarrow$, then this case is impossible, because the contractum of $\mathsf{q}_2\downarrow$ is a proper seq structure which cannot match the redex of $\mathsf{iq}_1\downarrow$.

□

PROPOSITION 5.52. *Rule* lis *permutes under any rule* $\rho \in \{\mathsf{ai}\downarrow, \mathsf{iq}_1\downarrow, \mathsf{liq}_3\downarrow, \mathsf{liq}_4\downarrow\}$ *if the redex of* lis *is not an active structure of the contractum of* ρ.

PROOF. We check the only case excluded by the conditions of Remark 5.49: The contractum of ρ is an active structure of the redex of lis. This is impossible because the contractum of the rule $\mathsf{ai}\downarrow$ is the unit and the contractum of $\mathsf{iq}_1\downarrow$, $\mathsf{liq}_3\downarrow$, and $\mathsf{liq}_4\downarrow$ is a seq structure which cannot be active structures inside the redex of lis. □

REMARK 5.53. *Despite the propositions above, it is impossible to obtain a partitioning of the provable* BV *structures within system* BVi *even in more general forms of Theorem 5.25, for instance given a derivation of the form* $\begin{array}{c}\langle K_1;K_2\rangle \\ \Delta\,\|_{\mathsf{BVsl}} \\ K\end{array}$ *we cannot obtain a derivation*

$$\begin{array}{c}[L_1,\ldots,L_m,\langle P_{1,1};P_{1,2}\rangle,\ldots,\langle P_{s,1};P_{s,2}\rangle,R_1,\ldots,R_n] \\ \|_{\mathsf{BVi}} \\ K\end{array},$$

such that there are derivations

$$\begin{array}{c}K_1 \\ \|_{\mathsf{BVsl}} \\ [L_1,\ldots,L_m,P_{1,1},\ldots,P_{s,1}]\end{array} \quad \textit{and} \quad \begin{array}{c}K_2 \\ \|_{\mathsf{BVsl}} \\ [P_{1,2},\ldots,P_{s,2},R_1,\ldots,R_n]\end{array}.$$

EXAMPLE 5.54. *Consider the following structure which is provable in system* BVi*:*

$$[\langle [\bar{a},\bar{b}];\bar{c};\bar{d}\rangle,\langle a;(\langle c;d\rangle,\bar{e})\rangle,\langle b;e\rangle]$$

We can apply Theorem 5.25 to this structure such that $\langle R;T\rangle = \langle [\bar{a},\bar{b}];\bar{c};\bar{d}\rangle$, *such that* $R = \langle [\bar{a},\bar{b}];\bar{c}\rangle$ *and* $T = \bar{d}$. *We get the derivation*

$$\begin{array}{c}\mathsf{ai}\downarrow \\ \mathsf{lis} \\ \mathsf{q}_1\downarrow\end{array} \dfrac{\dfrac{\dfrac{\langle [a,b];c;d\rangle}{\langle [a,b];(\langle c;d\rangle,[e,\bar{e}])\rangle}}{\langle [a,b];[(\langle c;d\rangle,\bar{e}),e]\rangle}}{[\langle a;(\langle c;d\rangle,\bar{e})\rangle,\langle b;e\rangle]}$$

such that $[R,\langle [a,b];c\rangle]$ *and* $[T,\bar{d}\,]$ *are provable in* BVsl. *A partitioning, even in the form of Remark 5.53 is impossible in system* BVi.

However, in the light of the observations above we can state the following conjecture:

CONJECTURE 5.55. *System* BV *and* BVi *are equivalent.*

It is immediate that every proof in BVi is also a proof in BV. However, transforming the proofs in BV into proofs in BVi is difficult: In the proofs of BV structures, because of the interleaving between the context management of the commutative copar operator, performed by the rule s, and the non-commutative seq operator, performed by the rule $\mathsf{q}{\downarrow}$, it is also impossible to decompose the proof into different phases [**Str03a**] such that at every phase different rules are applied. This is because the commutative and non-commutative context cooperate to promote the interactions described by the relation $\downarrow$, and then some instances of atomic interactions must be applied so that other substructures will be released so that a proof can be constructed:

EXAMPLE 5.56. *Straßburger gives the structure*

$$[\langle([d,\bar{d}],\langle a;b\rangle);c\rangle,\langle\bar{a};(\langle\bar{b};\bar{c}\rangle,[e,\bar{e}])\rangle]\,,$$

which is provable in BV*. This structure cannot be proved by constructing a proof bottom-up, by first applying only the rules* s *and* $\mathsf{q}{\downarrow}$*, and then only the rule* $\mathsf{ai}{\downarrow}$*.*

This complex behaviour is the source of difficulty also for the conjecture on the equivalence of system BV and pomset logic [**Gug07, Str03a**].

When we analyze system BV further, we observe that an important part of the nondeterminism in proof search is because of the rule $\mathsf{q}_2{\downarrow}$.

EXAMPLE 5.57. *Consider the following* BV *structure which is trivially provable in system* BVi *by applying the rule* $\mathsf{ai}{\downarrow}$ *twice:*

$$[a,\bar{a},b,\bar{b}]$$

The rule $\mathsf{q}_2{\downarrow}$ *can be applied to this structure in 50 different ways, e.g.,*

$$\mathsf{q}_2{\downarrow}\frac{\langle[a,b];[\bar{a},\bar{b}]\rangle}{[a,b,\bar{a},\bar{b}]}\quad,\quad \mathsf{q}_2{\downarrow}\frac{\langle a;[b,\bar{a},\bar{b}]\rangle}{[a,b,\bar{a},\bar{b}]}\quad,\quad \mathsf{q}_2{\downarrow}\frac{\langle b;[a,\bar{a},\bar{b}]\rangle}{[a,b,\bar{a},\bar{b}]}\quad,\ \textit{and}\ \ \mathsf{q}_2{\downarrow}\frac{\langle\bar{a};[a,b,\bar{b}]\rangle}{[a,b,\bar{a},\bar{b}]}\quad,$$

although this structure can be proved without any instance of this rule and the premise of only 15 of these 50 remain provable.

Unlike the rule switch 2, presented in Definition 4.24, which can be safely removed from system BV, the rule seq 2 cannot be removed from any system which is complete for provable BV structures. In order to see the reason for this consider the following example BV structure that I borrowed from [**Tiu01**]:

EXAMPLE 5.58. *The structure* $[\langle a;[b,c]\rangle,\langle[\bar{a},\bar{b}];\bar{c}\rangle]$ *cannot be proved without any instance of the rule* $\mathsf{q}_2{\downarrow}$*. Because there are no proper copar structures in this structure the rule* s *cannot be applied to this structure (without resorting to equations for unit). Furthermore, because of Proposition 5.9, application of any of the rules* $\mathsf{q}_1{\downarrow}$*,* $\mathsf{q}_3{\downarrow}$*, and* $\mathsf{q}_4{\downarrow}$ *results in a structure which is not provable. However, with the*

availability of the rule $\mathsf{q}_2{\downarrow}$*, among others, we have the following proof:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\circ}\circ{\downarrow}}{[b,\bar{b}]}\mathsf{ai}{\downarrow}}{[\langle[a,\bar{a}];b\rangle,\bar{b}]}\mathsf{ai}{\downarrow}}{[\langle a;b\rangle,\bar{a},\bar{b}]}\mathsf{liq}_3{\downarrow}}{\langle[\langle a;b\rangle,\bar{a},\bar{b}];[c,\bar{c}]\rangle}\mathsf{ai}{\downarrow}}{[\langle a;b;c\rangle,\langle[\bar{a},\bar{b}];\bar{c}\rangle]}\mathsf{iq}_1{\downarrow}}{[\langle a;[b,c]\rangle,\langle[\bar{a},\bar{b}];\bar{c}\rangle]}\mathsf{q}_2{\downarrow}$$

Definition 5.59. *Let system* BVu' *and* BVi'*, respectively, be the systems obtained by removing the rule* $\mathsf{q}_2{\downarrow}$ *from systems* BVu *and* BVi*, respectively.*

As it can be observed in Example 5.58, the systems BVu' and BVi' are not complete for provable BV structures, because these systems lack the rule $\mathsf{q}_2{\downarrow}$. However, the observations on the relationship between the commutative par relation and the non-commutative seq relation in relation webs makes it possible to state the conjecture below.

Definition 5.60. *Let* interaction seq 2 *be the rule*

$$\mathsf{iq}_2{\downarrow}\,\frac{S\langle R;T\rangle}{S[R,T]}$$

such that the following holds: Let μ *and* ν *be the sets of atom occurrences in structures* $\bar{R}$ *and* $\bar{T}$*, respectively. Then it holds that* $\mu \lhd \nu \subseteq \lhd_{S\{\ \}}$*.*

Definition 5.61. *Let system* $\mathsf{BVi}'' = \mathsf{BVi}' \cup \{\,\mathsf{iq}_2{\downarrow}\,\}$*.*

Conjecture 5.62. *The system* BVi'' *and system* BVi *are equivalent.*

The intuition behind this conjecture is the following: The role played by the rule $\mathsf{q}_2{\downarrow}$ in proof search is transforming the commutative par relation between two structures into the non-commutative seq relation. Although this is necessary in some cases, in others it is better not to allow the application of this rule if the duals of the atoms in these two structures are not already in a seq relation.

5.4. Cautious Rules

In a bottom-up application of the rules switch and seq in proof construction, besides promoting interactions between some atoms, the interaction between other atoms are broken as it can be seen in Example 5.10. However, if the structure being proved consists of pairwise distinct atoms, breaking the interaction between dual atoms in a bottom-up inference step delivers a structure which cannot be proved. The following definition introduces a further restriction on these inference rules that exploits this observation and allows only cautious instances of the inference rules which do not break the interaction between dual atoms.

Definition 5.63. Pruned switch *is the rule*

$$\mathsf{ps}\,\frac{S([R,W],T)}{S[(R,T),W]}\quad,$$

where $\mathsf{at}\,\overline{T} \cap \mathsf{at}\,W = \emptyset$*, and* pruned seq *is the rule*

$$\mathsf{pq}{\downarrow}\,\frac{S\langle [R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]}\quad ,$$

where $\mathsf{at}\,\overline{T} \cap \mathsf{at}\,U = \emptyset$ *and* $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,V = \emptyset$*.*

Definition 5.64. Pruned system BV*, or system* BVp *is the system given by* $\{\circ{\downarrow}, \mathsf{ai}{\downarrow}, \mathsf{ps}, \mathsf{pq}{\downarrow}\}$*.*

Proposition 5.65. *Let* P *be a* BV *structure that consists of pairwise distinct atoms and* Π *be a proof of* P *in* BV (BVs, BVsl*, respectively*)*. In* Π *all the instances of the rule* s (is, lis*, respectively*) *are instances of the rule* ps *and all the instances of the rule* $\mathsf{q}{\downarrow}$ *are instances of the rule* $\mathsf{pq}{\downarrow}$*.*

Proof. For any provable BV structure P, from Proposition 5.9, we have that for all the atoms $a \in \mathsf{at}\,P$, $(a,\bar{a}) \in {\downarrow_P}$. Thus, it suffices to show that the bottom-up application of the rules s and $\mathsf{q}{\downarrow}$ without respecting the restriction imposed by the rules ps and $\mathsf{pq}{\downarrow}$ result in structures that are not provable. Let P be a provable BV structure with pairwise distinct atoms such that

- $P = S[(R, T\{a\}), W\{\bar{a}\}]$, that is, $\mathsf{at}\,\overline{W\{\bar{a}\}} \cap \mathsf{at}\,T\{a\} \supseteq \{a\}$. Applying the rule s without the restriction imposed by the rule ps results in the structure $P' = S([R, W\{\bar{a}\}], T\{a\})$. It follows that $(a,\bar{a}) \notin {\downarrow_{P'}}$ which contradicts with Proposition 5.9.

- $P = S[\langle R; U\{a\}\rangle, \langle T\{\bar{a}\}; V\rangle]$, that is, $\mathsf{at}\,\overline{T\{\bar{a}\}} \cap \mathsf{at}\,U\{a\} \supseteq \{a\}$. Applying the rule $\mathsf{q}{\downarrow}$ without the restriction imposed by the rule $\mathsf{pq}{\downarrow}$ results in the structure $P' = S\langle [R, T\{\bar{a}\}]; [U\{a\}, V]\rangle$. It follows that $(a,\bar{a}) \notin {\downarrow_{P'}}$ which contradicts with Proposition 5.9.

□

Proposition 5.66. *Let* P *be a* BV *structure that consists of pairwise distinct atoms and* Π *be a proof of* P *in* BVi*. In* Π *all the instances of the rule* s *are instances of the rule* ps *and all the instances of the rule* $\mathsf{iq}_1{\downarrow}$,$\mathsf{q}_2{\downarrow}$, $\mathsf{liq}_3{\downarrow}$*, and* $\mathsf{liq}_4{\downarrow}$ *are instances of the rule* $\mathsf{pq}{\downarrow}$*.*

Proof. Follows immediately from Remark 5.43 and Proposition 5.65. □

5.5. Implementation in Maude

In Chapter 3, we have seen that the bottom up application of an inference rule can be represented as a rewriting rule that rewrites the conclusion to the premise of the inference rule. Similarly, inference rules with conditions can be represented as conditional rewrite rules. For instance, consider the following rewrite rule for the inference rule interaction seq rule 1:

$$\mathsf{iq}_1{\downarrow}:\ [\langle R;U\rangle,\langle T;V\rangle] \ \rightarrow\ \langle [R,T];[U,V]\rangle \quad \text{if}\ \ \mathsf{at}\,\overline{R} \cap \mathsf{at}\,T \neq \emptyset \ \wedge\ \mathsf{at}\,\overline{U} \cap \mathsf{at}\,V \neq \emptyset$$

The inference rules of system BVi, that impose restrictions on the structures, can be implemented in the language Maude, by considering these inference rules as such conditional rewrite rules. The conditional rewrite rules are defined by the keyword `crl` in their Maude representation with the syntax

```
  crl [<Label>] : <Term-1> => <Term-2>
                       if <Condition-1> /\ ... /\ <Condition-k> .
```

where the conditions can be equations which are computed by a functional module. The implementation below exploits these features of the language Maude for implementing systems FBVi and BVi. The functional module `Can-interact` contains the equations that implement the conditions of the inference rules of system BVi. When we remove the rewrite rules for the rule $\mathsf{q}\downarrow$ from the module for system BVi below, we obtain an implementation of system FBVi.

```
fmod BV-Signature is

  sorts Atom Unit Structure .
  subsort Atom < Structure .
  subsort Unit < Structure .

  ops a b c d e f g h i j k l m n p q r s  : -> Atom .

  op o       : -> Unit .
  op -_      : Atom -> Atom [ prec 50 ].
  op -_      : Structure -> Structure [ prec 50 ] .
  op [_,_]   : Structure Structure -> Structure [assoc comm] .
  op {_,_}   : Structure Structure  -> Structure [assoc comm] .
  op <_;_>   : Structure Structure -> Structure [assoc] .
endfm

fmod Can-interact is
  inc BV-Signature .

  sort Interaction_Query .
  op  can-interact   : -> Interaction_Query .
  op  empty-set : -> Interaction_Query .

  op   _or_ : Interaction_Query Interaction_Query
                      -> Interaction_Query [assoc comm prec 70] .

  op   _ci_ : Atom Structure -> Interaction_Query [prec 60] .


  var R T U V : Structure .
  var A B     : Atom .
  var C       : Interaction_Query .

  eq    A  ci - A = can-interact .
  eq  - A  ci   A = can-interact .
  eq    A  ci   B = empty-set [owise] .

  eq [ T , U ] ci R =  T ci R  or  U ci R .
  eq { T , U } ci R =  T ci R  or  U ci R .
  eq < T ; U > ci R =  T ci R  or  U ci R .
```

```
  eq A ci [ R , T ] =  A ci R  or  A ci T .
  eq A ci { R , T } =  A ci R  or  A ci T .
  eq A ci < R ; T > =  A ci R  or  A ci T .

  eq can-interact or  C  = can-interact .
  eq empty-set    or  C  = C .
endfm

mod BVi is
   inc Can-interact .

   var R T U V P Q : Structure .    var A : Atom .

   rl [ai-u1-down]     : [ [ A , - A ] , R ]       => R .
   rl [ai-u2-down]     : { [ A , - A ] , R }       => R .
   rl [ai-u3-down]     : < [ A , - A ] ; R >       => R .
   rl [ai-u4-down]     : < R ; [ A , - A ] >       => R .

   crl  [rls1] : [ { R , T } , A ] =>
                              { [ R , A ] , T }
                                    if R ci A = can-interact .

   crl  [rls2] : [ { R , T } , { U, V } ] =>
                             { [ R , { U , V } ] , T }
                             if R ci  { U , V } = can-interact .

   crl [q1-down] : [ < R ; T > , < U ; V > ] =>
                             < [R,U] ; [T,V] >
                                  if  R ci U = can-interact  /\
                                      T ci V = can-interact .

   rl [q2-down] : [ R , T ] => < R ; T >  .

   crl [q31-down] : [ A , < R ; T > ]  => < [ R , A ] ; T >
                                       if R ci A = can-interact .

   crl [q32-down] : [ { U , V } , < R ; T > ] =>
                                  < [ R , { U , V } ] ; T >
                               if R ci { U , V } = can-interact .

   crl [q33-down] : [ < U ; V > , < R ; T > ] =>
                             < [ R , < U ; V > ] ; T >
                               if R ci < U ; V > = can-interact .

   crl [q41-down] : [ A , < R ; T > ]  => < R ; [ T , A ] >
                                       if T ci A = can-interact .
```

```
crl [q42-down] : [ { U , V } , < R ; T > ] =>
                              < R ; [ T , { U , V } ] >
                               if T ci { U , V }  = can-interact .

crl [q43-down] : [ < U ; V > , < R ; T > ] =>
                              < R ; [ T , < U ; V > ] >
                                if T ci < U ; V > = can-interact .
endm
```

Some representative examples of experiments for comparing the performance of systems FBV and FBVi are as follows: Consider the following provable flat BV structures in the implementation in the context of system FBV.

1. $[a, b, (\bar{a}, \bar{c}), (\bar{b}, c)]$
2. $[a, b, (\bar{a}, \bar{b}, [a, b, (\bar{a}, \bar{b})])]$
3. $[a, b, (\bar{a}, \bar{b}, [c, d, (\bar{c}, \bar{d})])]$
4. $[a, b, (\bar{a}, \bar{b}, [c, d, (\bar{c}, \bar{d}, [e, f, (\bar{e}, \bar{f})])])]$

Let us call the system FBVu the system obtained by removing the inference rules $\mathsf{q}_1{\downarrow}$, $\mathsf{q}_2{\downarrow}$, $\mathsf{q}_3{\downarrow}$, and $\mathsf{q}_4{\downarrow}$ from system BVu. When we search for a proof of these queries within the Maude modules for the systems FBVu and FBVi, we get the results in Table 5.1.

		finds a proof		search terminates	
Query	System	# states explored	finds a proof in # ms (cpu)	# states explored	finds a proof in # ms (cpu)
1.	FBVu	342	60	369	100
	FBVi	34	10	44	20
2.	FBVu	1041	100	1074	100
	FBVi	264	0	318	10
3.	FBVu	1671	310	1759	370
	FBVi	140	0	146	10
4.	FBVu	–	–	–	–
	FBVi	6595	1370	6690	1420

TABLE 5.1. Representative performance comparison of proof search in the implementations of the systems FBV, and FBVi. The search on Query 4. halted by running out of memory after having spent approximately 3GB memory and 80 minutes (cpu).

In the experiments presented in Table 5.1, it is important to observe that the number of explored states is proportional with the time spent for finding a proof.

Table 5.2 gives a performance comparison of the implementations of the other systems that I discussed so far in this chapter with the performance of the system BVu. These experiments were performed on the structures in Section 4.3.1 which were used for the experiments presented in Table 4.1.

It is important to note that the proof search strategy used in these implementations is breadth-first search. This search strategy provides a complete exploration of the search space. However, in proof search, the size of the proof search space expands rather quickly after a small number of steps in the depth of the search

tree. For instance, if at every node there are in average 10 (there are often many more) different possible rule instances, the search space which admits a proof with length n has

$$10^1 + 10^2 + \cdots + 10^n$$

nodes, which should be visited, in order to reach this shortest proof. The problem persists with the depth-first strategy: If the node delivering the proof is at the right-most node of depth n, and the algorithm starts exploring the search space from the left-most nodes, then even many more nodes must be visited, before the proof is found.

As we have seen in Example 5.57, the main source of nondeterminism in system BVi is the rule $\mathsf{q}_2{\downarrow}$. This can be observed also in the results of the experiments shown in Table 5.2 . Redesigning this rule in such a way that gets rid of the unnecessary nondeterminism, possibly as described in Definition 5.60, would provide a much better performance in proof search.

Because system BV is a multiplicative logic, the essential nondeterminism in system BV is due to the multiple occurrence of the same atom in the structure whose proof is being searched. In other words, deciding which atom a to pair with which atom $\bar{a}$ is the main source of nondeterminism in these systems.

EXAMPLE 5.67. *Consider the* FBV *structure*

$$[a, \bar{a}, (a, \bar{a})]$$

which does not consist of pairwise distinct atoms. The following structures

$$[a_1, \bar{a}_1, (a_2, \bar{a}_2)] \qquad [a_1, \bar{a}_2, (a_2, \bar{a}_1)]$$

are obtained from the structures above by renaming the atoms. The first structure is not provable in FBV *because the atoms* a_2 *and* $\bar{a}_2$ *are not in a* $\downarrow$ *relation, as it can be seen in Proposition 5.9. However, the second one is provable in system* FBVi.

In [**Gue99**], Guerrini has shown that correctness of a multiplicative proof net [**Gir87**] can be performed in linear time. This result implies that provability of multiplicative linear logic structures that consist of pairwise distinct atoms can be performed in linear time. Because system BV is a multiplicative logic, it is plausible to argue that the ideas of [**Gue99**] can be carried to system BV. In fact, logical expressions of pomset logic [**Ret97, Ret99**], which is a logic similar logic to system BV, admit a graphical representation called R&B-cographs resembling proof nets. R&B-cographs enjoy a correctness criterion. Guglielmi [**Gug07**] and Straßburger [**Str03a**] conjectured that system BV and pomset logic are equivalent.

5.6. Nondeterminism in Classical Logic

Systems in the calculus of structures follow a common scheme where the context management of commutative operators is performed by the switch rule. System KSg for classical logic is no exception to this. In this section, I will show that, similar to system BV, the switch rule of system KSg can be safely replaced with the lazy interaction switch rule in order to reduce nondeterminism in proof search. I will then show that this technique is complete also for system KS, which is the local system for classical logic in the calculus of structures [**Brü03b**].

DEFINITION 5.68. *The system* KSgi *is the system obtained from system* KSg *by replacing the rule* s *with the rule* lis. *System* KSgi *is defined on* KSg *structures.*

		finds a proof		search terminates	
Query	System	in # millisec.	after # rewrites	in # millisec.	after # rewrites
1.	BVu	80	7034	90	7929
	BVi	60	12210	60	12210
2.	BVu	150	14684	670	57617
	BVi	20	10390	180	40227
	BVu′	10	1108	20	1531
	BVi′	0	168	0	785
3.	BVu	330	27252	780	57956
	BVi	120	28122	230	36452
	BVu′	40	2636	50	3434
	BVi′	30	6000	40	6562
4.	BVu	390	36549	1370	117427
	BVi	140	40365	520	95408
	BVu′	30	2073	40	2673
	BVi′	10	2382	20	3070
5.	BVu	460	43304	1060	86880
	BVi	160	36334	410	65744
	BVu′	10	792	10	830
	BVi′	0	862	0	910
6.	BVu	3270	258313	6140	447774
	BVi	550	158375	1090	224313
	BVu′	330	20878	410	24493
	BVi′	70	24304	100	29225
7.	BVu	4420	382911	7280	605438
	BVi	900	232448	2010	415960
8.	BVu	6200	469793	19060	1248859
	BVi	1270	477279	3200	832513
	BVu′	620	43031	910	57301
	BVi′	80	28154	90	31589
9.	BVu	9520	855145	18370	1568647
	BVi	2830	1285032	3990	1602574
	BVu′	1320	102154	2050	147409
	BVi′	120	49467	170	57458
10.	BVu	17660	1187905	70680	3129460
	BVi	3670	1495887	8470	2306500
	BVu′	1630	116382	2240	144178
	BVi′	210	77704	260	86538

TABLE 5.2. Representative performance comparison of proof search in the implementations of the systems BVu, BVi, BVu′ and BVi′. Queries 1 and 7 are not provable in systems BVu′ and BVi′.

In the following, I will show that the systems KSg and KSgi are equivalent. For this purpose, I will first collect some definitions and lemmas that will be necessary.

LEMMA 5.69. *The rule* $\mathsf{w}\!\downarrow$ *of system* KSg *permutes under the rule* s.

PROOF. It suffices to check the cases excluded by Remark 5.49.

(a) The redex is of $\mathsf{w}\!\downarrow$ is inside an active structure of the contractum of s. In this case we permute as follows:

$$\dfrac{\dfrac{S(\mathsf{ff},T)}{S([R,U],T)}\,\mathsf{w}\!\downarrow}{S[(R,T),U]}\,\mathsf{s} \qquad \rightsquigarrow \qquad \dfrac{\dfrac{S(\mathsf{ff},T)}{S(R,T)}\,\mathsf{w}\!\downarrow}{S[(R,T),U]}\,\mathsf{w}\!\downarrow$$

(b) The contractum of s is inside an active structure of the redex of $\mathsf{w}\!\downarrow$. All the cases being analogous to below, in this case we permute as follows:

$$\dfrac{\dfrac{S([R,U],\mathsf{ff})}{S([R,U],T,P)}\,\mathsf{w}\!\downarrow}{S([(R,T),U],P)}\,\mathsf{s} \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\dfrac{S([R,U],\mathsf{ff})}{S([(R,\mathsf{ff}),U],\mathsf{ff})}\,\mathsf{s}}{S([(R,T),U],\mathsf{ff})}\,\mathsf{w}\!\downarrow}{S([(R,T),U],P)}\,\mathsf{w}\!\downarrow$$

□

THEOREM 5.70. *A structure* R *has a proof in* KSg *if and only if there are structures* R_1, R_2, *and* R_3 *and there is a proof such that*

$$\begin{array}{l} \Vdots\,\{\mathsf{ai}\!\downarrow\} \\ R_3 \\ \shortparallel\,\{\mathsf{s}\} \\ R_2 \\ \shortparallel\,\{\mathsf{w}\!\downarrow\} \\ R_1 \\ \shortparallel\,\{\mathsf{c}\!\downarrow\} \\ R \end{array} .$$

PROOF. The only if direction being trivial, let us see the proof of the if direction: From Theorem 4.56 it follows that if R has a proof in KSg, then there is a proof of the following form where R_2 is in conjunctive normal form:

$$\begin{array}{l} \Vdots\,\{\mathsf{ai}\!\downarrow\} \\ R_3 \\ \shortparallel\,\{\mathsf{w}\!\downarrow\} \\ R_2 \\ \Delta\,\shortparallel\,\{\mathsf{s},\mathsf{c}\!\downarrow\} \\ R \end{array}$$

Further, from Lemma 5.69, we know that the rule s permutes over the rule $\mathsf{w}\!\downarrow$. Thus, it suffices to show that we can replace the derivation Δ in the derivation above as follows, because we can then permute all the instances of the rule s over all the instances of $\mathsf{w}\!\downarrow$ by using Lemma 5.69.

$$\begin{array}{l} R_2 \\ \Delta\,\Vdots\,\{\mathsf{s},\mathsf{c}\!\downarrow\} \\ R \end{array} \qquad \rightsquigarrow \qquad \begin{array}{l} R_2 \\ \Vdots\,\{\mathsf{s}\} \\ R_1 \\ \Vdots\,\{\mathsf{c}\!\downarrow\} \\ R \end{array}$$

Let us prove this with structural induction on R. If R is an atom or the unit tt or ff, then it is already in conjunctive normal form. If $R = (T, U)$ or $R = [T, U]$ then we have the following derivations by induction hypothesis

$$\begin{array}{cc} T_2 & U_2 \\ {\scriptstyle \Delta'_T} \,\|\!|\, {\scriptstyle \{\mathsf{s}\}} & {\scriptstyle \Delta'_U} \,\|\!|\, {\scriptstyle \{\mathsf{s}\}} \\ T_1 & U_1 \\ {\scriptstyle \Delta_T} \,\|\!|\, {\scriptstyle \{\mathsf{c}\downarrow\}} & {\scriptstyle \Delta_U} \,\|\!|\, {\scriptstyle \{\mathsf{c}\downarrow\}} \\ T & U \end{array}$$

where T_2 and U_2 are in conjunctive normal form. Let n be the number of disjunctions in U_2. We assume that n is greater than one. Otherwise, we can exchange T_2 with U_2, or if in both T_2 and U_2, there are less than 2 disjunctions, then they would be already in conjunctive normal form. We construct the derivations we need for $R = (T, U)$ and $R = [T, U]$, respectively, as follows:

$$\begin{array}{cc} & R_2 \\ & \|\!|\, {\scriptstyle \{\mathsf{s}\}} \\ (T_2, U_2) & [T_2, \ldots, T_2, U_2] \\ {\scriptstyle [\Delta'_T, \Delta'_U]} \,\|\!|\, {\scriptstyle \{\mathsf{s}\}} & {\scriptstyle [\Delta'_T, \ldots, \Delta'_T, \Delta'_U]} \,\|\!|\, {\scriptstyle \{\mathsf{s}\}} \\ (T_1, U_1) & [T_1, \ldots, T_1, U_1] \\ {\scriptstyle [\Delta_T, \Delta_U]} \,\|\!|\, {\scriptstyle \{\mathsf{c}\downarrow\}} & {\scriptstyle [\Delta_T, \ldots, \Delta_T, \Delta_U]} \,\|\!|\, {\scriptstyle \{\mathsf{c}\downarrow\}} \\ (T, U) & [T, \ldots, T, U] \\ & \|\!|\, {\scriptstyle \{\mathsf{c}\downarrow\}} \\ & [T, U] \end{array}$$

□

Definition 5.71. *The system* Ki *is the system obtained from system* KSg *by replacing the rule* s *with the rule* lis *and removing the rules* w↓ *and* c↓. *System* Ki *is defined on* KSg *structures.*

The reader might realize that there is a significant similarity between the systems Ki and the system FBVi (FBV) (the system for multiplicative linear logic extended by the rules mix and nullary mix). Indeed, these two systems have the same set of inference rules. However, the treatment of the units in these systems is quite different: In system FBV there is a single unit, which is shared by all the connectives. On the other hand, in system Ki, there are two different units, tt and ff, which are units for different operators. If we consider the multiplicative fragment of the linear logic system LS (where there are two different units 1 and $\bot$, and mix and nullary mix are not valid), the similarity with system Ki is greater. However, there is a significant difference between this system and system Ki: In system KSg, thus also in system Ki, the equalities $\mathsf{ff} = (\mathsf{ff}, \mathsf{ff})$ and $\mathsf{tt} = [\mathsf{tt}, \mathsf{tt}]$ hold. However, in multiplicative linear logic, the analogs of these equalities do not hold, and they are not derivable.

Below, I will carry some definitions and results from the Section 5.2 to system Ki:

DEFINITION 5.72. *Let R, T be* KSg *structures such that $R \neq \mathsf{ff} \neq T$. R and T are* independent for Ki *if and only if*

$$\overset{\mathsf{Ki}}{\underset{[R,T]}{\Vdash}} \quad \textit{implies} \quad \overset{\mathsf{Ki}}{\underset{R}{\Vdash}} \quad \textit{and} \quad \overset{\mathsf{Ki}}{\underset{T}{\Vdash}} \quad .$$

Otherwise, they are dependent.

PROPOSITION 5.73. *For any* Ki *structures R and T, if $\mathsf{at}\bar{R} \cap \mathsf{at}T = \emptyset$ then R and T are independent.*

PROOF. Analogous to the proof of Proposition 5.23: Construct a proof of R by replacing all the substructures of T in Π with ff. Similarly, construct a proof of T by replacing all the substructures of R in Π with ff. □

LEMMA 5.74.

If $\Pi \overset{\mathsf{Ki}}{\underset{[P,U]}{\Vdash}}$ *then, for any structure R, there is a derivation* $\overset{R}{\underset{[(R,P),U]}{\Vdash^{\mathsf{Ki}}}}$.

PROOF. Analogous to the proof of Lemma 5.24. □

THEOREM 5.75. *(Shallow splitting for* Ki*) For all structures R, T, and P, if $[(R,T),P]$ is provable in* Ki *then there exists P_1, P_2 and* $\overset{[P_1,P_2]}{\underset{P}{\Delta\Vdash^{\mathsf{Ki}}}}$ *such that $[R,P_1]$ and $[T,P_2]$ are provable in* Ki.

PROOF. Proof by induction, similar to the proof of Theorem 5.25. Consider the following statement, where the relation $\downarrow_R$ for a structure R is defined as for BV structures with the difference that the occurrence of the units tt and ff are not considered in $\downarrow_R$.

$$\begin{aligned}\mathsf{C}(n) = \quad & \forall n'.\forall R,T,P.\Big((n' \leq n \\ & \wedge \quad n' = |\downarrow_{[(R,T),P]}| \\ & \wedge \quad \text{there is a proof } \overset{}{\underset{[(R,T),P]}{\Vdash}} \ ) \\ & \Rightarrow \ \exists P_1,P_2.(\ \overset{[P_1,P_2]}{\underset{P}{\Vdash}} \ \wedge \ \overset{}{\underset{[R,P_1]}{\Vdash}} \ \wedge \ \overset{}{\underset{[T,P_2]}{\Vdash}} \ )\Big) .\end{aligned}$$

The statement of the theorem is equivalent to $\forall n.\mathsf{C}(n)$ and the proof is done by taking n as the induction measure. The base case is trivial. For the inductive cases, let us always assume $\mathsf{tt} \neq P \neq \mathsf{ff}$, because when this is not the case the theorem is trivially proved. For the same reason, I assume $R \neq \mathsf{tt} \neq T$ and $R \neq \mathsf{ff} \neq T$. Consider the bottom rule instance of the proof Π of $[(R,T),P]$:

$$\rho\,\frac{\overset{}{\underset{Q}{\Vdash}}}{\overset{}{\underset{[(R,T),P]}{\Vdash}}} \quad ,$$

I assume that ρ is non-trivial, because every proof with trivial rule instances can be rewritten as a proof where these trivial instances are removed. The cases for $\rho = \mathsf{ai}{\downarrow}$ and $\rho = \mathsf{lis}$, respectively, are as in the Case 2.a and Case 2.c of Theorem 5.25, respectively, by using Lemma 5.74 instead of Lemma 5.24. □

THEOREM 5.76. (Context reduction for Ki) *For all structures R and for all contexts $S\{\ \}$ such that $S\{R\}$ is provable in Ki, there exists a structure U such that for all structures X there exist derivations:*

$$\begin{array}{c}[X,U]\\ \shortparallel\mathsf{Ki}\\ S\{X\}\end{array} \qquad \textit{and} \qquad \begin{array}{c}\vdots\\ \shortparallel\mathsf{Ki}\\ [R,U]\end{array} .$$

PROOF. Similar to the proof of Theorem 5.29 by induction on the size of $S\{\mathsf{ff}\}$. The base case is trivial: $U = \mathsf{ff}$. There are two inductive cases:

(1) $S\{\ \} = (S'\{\ \}, P)$, for some $P \neq \mathsf{tt}$. There must be proofs in Ki of $S'\{R\}$ and of P, thus it must be that $P \neq \mathsf{ff}$. By applying the induction hypothesis, we can find U and construct, for all X:

$$\begin{array}{c}[X,U]\\ \shortparallel\mathsf{Ki}\\ S'\{X\}\\ \shortparallel\mathsf{Ki}\\ (S'\{X\}, P)\end{array}$$

such that $[R,U]$ is provable in Ki.

(2) $S\{\ \} = [S'\{\ \}, P]$, for some $P \neq \mathsf{ff}$ such that $S'\{\ \}$ is not a proper par: If $P = \mathsf{tt}$ or $S'\{\,\mathsf{ff}\,\} = \mathsf{ff}$ then the theorem is proved; otherwise it must be that $S'\{\ \} = (S''\{\ \}, P')$, for some $P \neq \mathsf{tt}$. The rest is same as in Case 2.a of the proof of Theorem 5.29, by using Lemma 5.74 instead of Lemma 5.24.

□

We can now state the main result of this section:

THEOREM 5.77. *System* KSg *and* KSgi *are equivalent.*

PROOF. Observe that every proof in KSgi is also a proof in KSg. For the other direction, let R be a structure which has a proof in KSg. From Theorem 5.70, we have the following proof:

$$\begin{array}{c}\vdots\ \Pi\\ \shortparallel\{\mathsf{s},\mathsf{ai}{\downarrow}\}\\ R'\\ \vdots\ \Delta\\ \shortparallel\{\mathsf{w}{\downarrow},\mathsf{c}{\downarrow}\}\\ R\end{array}$$

Replace the proof Π with a proof in Ki, analogous to the proof of Theorem 5.32, by using Theorem 5.76, Theorem 5.75, and Lemma 5.74. □

5.6.1. Nondeterminism in a Local System for Classical Logic. System KS [**Brü03b**] is a local system for classical logic. System KS is obtained from system KSg by replacing the weakening rule with the *atomic weakening* rule, and the contraction rule with the *atomic contraction* rule and another rule, called the *medial*:

DEFINITION 5.78. *The following rules are called* atomic weakening *(*$\mathsf{aw}\downarrow$*),* atomic contraction *(*$\mathsf{ac}\downarrow$*) and* medial *(*m*), respectively:*

$$\mathsf{aw}\downarrow \frac{S\{\mathsf{ff}\}}{S\{a\}} \qquad \mathsf{ac}\downarrow \frac{S[a,a]}{S\{a\}} \qquad \mathsf{m}\,\frac{S[(R,U),(T,V)]}{S([R,T],[U,V])}$$

DEFINITION 5.79. *System* KS *is the system obtained from system* KSg *by replacing the rule* $\mathsf{w}\downarrow$ *and the rule* $\mathsf{c}\downarrow$ *with the rules* $\mathsf{aw}\downarrow$, $\mathsf{ac}\downarrow$ *and* m.

DEFINITION 5.80. *The following rule is called* tt*-weakening (*$\mathsf{ttw}\downarrow$*):*

$$\mathsf{ttw}\downarrow \frac{S\{\mathsf{ff}\}}{S\{\mathsf{tt}\}}$$

PROPOSITION 5.81. *The rule* $\mathsf{ttw}\downarrow$ *is derivable for* $\{\mathsf{s}\}$.

PROOF. Take the following derivation:

$$\begin{array}{c} \approx \dfrac{S\{\mathsf{ff}\}}{S([\mathsf{tt},\mathsf{tt}],\mathsf{ff})} \\ \mathsf{s}\,\dfrac{}{} \\ \approx \dfrac{S[(\mathsf{tt},\mathsf{ff}),\mathsf{tt}]}{S\{\mathsf{tt}\}} \end{array}$$

□

Brünnler and Tiu proved the following two theorems in [**BT01**].

THEOREM 5.82. *The rule* $\mathsf{w}\downarrow$ *is derivable for* $\{\mathsf{aw}\downarrow, \mathsf{ttw}\downarrow\}$. *The rule* $\mathsf{c}\downarrow$ *is derivable for* $\{\mathsf{ac}\downarrow, \mathsf{m}\}$.

THEOREM 5.83. *System* KS *and* KSg *are equivalent.*

DEFINITION 5.84. *The system* KSi *is the system obtained from system* KS *by replacing the rule* s *with the rules* lis *and* $\mathsf{ttw}\downarrow$.

COROLLARY 5.85. *Systems* KSi, KSgi, KS *and* KSg *are equivalent.*

PROOF. Follows immediately from Theorem 5.70, Theorem 5.77, Theorem 5.82 and Theorem 5.83. □

5.7. Discussion

In this chapter, I have introduced a technique for reducing nondeterminism in proof search by restricting the application of the inference rules. The inference rules for context management, which are redesigned with respect to this technique, can be applied only in certain ways that promote the interaction, in the sense of a specific mutual relation between dual atoms. Because proofs are constructed by annihilating dual atoms, the restrictions on the applications of the inference rules do not only reduce the breadth of the search space drastically, but also make the shorter proofs more immediately accessible.

The mutual relationships, that I used, originate from a graphical representation (relation webs) of BV structures. However, we have seen that the intuition provided by these relations can be analogously carried to other logics such as classical logic. By using this technique, I obtained a class of equivalent systems to system BV where nondeterminism is reduced at different levels. Then I demonstrated that this technique does not depend on the multiplicative nature of system BV: It can be analogously applied to systems for classical logic, i.e., systems KSg and KS, in a way that results in equivalent systems to these systems, where nondeterminism is reduced.

The splitting argument that I used in the completeness proofs of the resulting systems was initially invented as a technique in [**Gug07**] for proving cut-elimination. For system BVsl, I employed a simple specialization of the splitting theorem in [**Gug07**]. Because the procedure for showing the completeness of these systems is closely related with a cut-elimination procedure, the new systems which are obtained by means of this new technique remain clean from a proof theoretic point of view. Further, because splitting provides a partitioning of the structure being proved, it can also be used as a search strategy in conjunction with the technique of this thesis.

In [**Str03a**], Straßburger used the splitting technique to prove cut elimination in a linear logic system in the calculus of structures. In [**GS02, Str03a**], decomposition and splitting are used together to prove cut-elimination for system NEL in a similar way to the completeness proof of system KSgi in this chapter. Furthermore, all the systems in the calculus of structures follow a scheme where the context management is performed by the *switch* rule, which is common to all systems. Because system BVsl is obtained by replacing the switch rule with the *restricted lazy switch rule* by means of splitting, I believe that the methods that I presented in this chapter can be generalized to systems NEL, LS, and other systems in the calculus of structures.

CHAPTER 6

System BV is NP-complete

Since its emergence, the multiplicative fragment of linear logic remained in the focus of researchers due to its resource conscious features that capture properties of concurrent computation (see, e.g., [**Bel97**]). Max Kanovich shows in [**Kan91, Kan92**] that multiplicative linear logic (MLL) is NP-complete. In [**LW94**], Lincoln and Winkler show that constant-only fragment of MLL is also NP-complete. However, from the point of view of applications, multiplicative linear logic lacks a natural notion of sequentiality, which is crucial for expressing many computational phenomena, e.g., sequential composition of processes in concurrency theory. System BV extends MLL with the rules *mix* (mix), *nullary mix* (mix0), and a self-dual non-commutative logical operator *seq*. Thus, system BV extends the applications of MLL to those where sequential composition is crucial.

System NEL extends system BV with the exponentials of linear logic. In other words, system NEL is an extension of multiplicative exponential linear logic (MELL) with the rules mix, mix0, and the self-dual non-commutative logical operator seq. Although it is unknown whether multiplicative exponential linear logic is decidable or not, in [**Str03c**], Straßburger showed that system NEL is undecidable. Figure 2.5 summarizes the relationship between MLL, FBV, BV, MELL, and NEL. In this chapter, I will show that when MLL is extended with mix and mix0, it remains NP-complete. Then I will show that the decision problem for system BV is also NP-complete. For this purpose, I will resort to some results from Chapter 5, which will serve as combinatoric proof theoretic tools.

6.1. System BV is NP-hard

In this section, I present an encoding of the 3-Partition Problem [**GJ79**] in system FBV to show the NP-hardness of this logic and system BV. This problem was also used by Lincoln and Winkler, in [**LW94**], to show the NP-hardness of the constant only fragment of MLL. By providing a similar encoding, and resorting to the proof theory of system FBV, I will provide a very simple correctness proof without going into a complicated case analysis.

PROBLEM 6.1. *(3-Partition) Given a set of* $A = \{a_1, a_2, \ldots, a_{3m}\}$ *of elements, a bound* $B \in \mathbb{N}^+$*, and a size* $S(a) \in \mathbb{N}^+$ *for each* $a \in A$ *such that* $\frac{1}{4}B < S(a) < \frac{1}{2}B$ *and* $\sum_{a \in A} S(a) = Bm$*, does there exist a partition of* A *into* m *disjoint subsets* A_i *so that* $\sum_{a \in A_i} S(a) = B$ *for each* A_i *in the partition.*

The constraints on the $S(a)$ imply that such a partition must have exactly three elements in each of its sets. This problem is NP-complete in the strong sense, which implies that even when the input is represented in unary, the problem is NP-hard. This property of 3-Partition is essential for my encoding, because I represent the input problem by using atoms.

6.1.1. Encoding the 3-Partition Problem in FBV. Given an instance of 3-Partition equipped with a set $A = \{a_1, a_2, \ldots, a_{3m}\}$, a unary function S, and a natural number B, presented as a tuple $\langle A, m, B, S\rangle$, the encoding function θ is defined as $\theta(\langle A, m, B, S\rangle) =$

$$[(k, [\underbrace{c, \ldots, c}_{\times S(a_1)}]), \ldots, (k, [\underbrace{c, \ldots, c}_{\times S(a_{3m})}]), \underbrace{([\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})], \ldots, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])}_{\times m}]$$

LEMMA 6.2. *Let $S(a_1)$, $S(a_2)$ and $S(a_3)$ be natural numbers such that, for some natural number B, it holds that $\frac{1}{4}B < S(a_1), S(a_2), S(a_3) < \frac{1}{2}B$. If $S(a_1)+S(a_2)+S(a_3) = B$, then*

$$\begin{array}{c} [R, Q] \\ \Delta \Big\| \mathsf{FBV} \\ [R, (k, [\underbrace{c, \ldots, c}_{\times S(a_1)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_2)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_3)}]), (Q, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])] \end{array}.$$

PROOF. Take the following derivation where the redex in the conclusion of the applied rule is highlighted.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[R, (Q, [\underbrace{c, \ldots, c}_{\times S(a_1)}, \underbrace{c, \ldots, c}_{\times S(a_2)}, \underbrace{c, \ldots, c}_{\times S(a_3)}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])]}\,\mathsf{i}\downarrow \quad [R, Q]}{\vdots}\,\mathsf{ai}\downarrow}{[R, (k, [c, \ldots, c]), (k, [c, \ldots, c]), (Q, [\bar{k}, \bar{k}, c, \ldots, c, (\bar{c}, \ldots, \bar{c})])]}\,\mathsf{s}}{[R, (k, [c, \ldots, c]), (k, [c, \ldots, c]), (Q, [([k, \bar{k}], [c, \ldots, c]), \bar{k}, \bar{k}, (\bar{c}, \ldots, \bar{c})])]}\,\mathsf{ai}\downarrow}{[R, (k, [c, \ldots, c]), (k, [c, \ldots, c]), (Q, [(k, [c, \ldots, c]), \bar{k}, \bar{k}, \bar{k}, (\bar{c}, \ldots, \bar{c})])]}\,\mathsf{s}}{[R, (k, [\underbrace{c, \ldots, c}_{\times S(a_1)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_2)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_3)}]), (Q, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])]}\,\mathsf{s}$$

□

THEOREM 6.3. *If a 3-Partition problem $\langle A, m, B, S\rangle$ is solvable, then there is a proof of $\theta(\langle A, m, B, S\rangle)$ in* FBV.

PROOF. By induction on m: The base case is given by the proof consisting of the rule $\circ\downarrow$. For the inductive case, assume that the result holds for $m = k$. Assume that the problem $\langle A \cup \{a_1, a_2, a_3\}, k+1, B, S\rangle$ is solvable such that $\{a_1, a_2, a_3\}$ is a 3-partition in the solution. It follows that there is a solvable 3-Partition problem given with $\langle A, k, B, S\rangle$. Let $[R, Q] = \theta(\langle A, k, B, S\rangle)$. Take the proof below of $\theta(\langle A \cup \{a_1, a_2, a_3\}, k+1, B, S\rangle)$ where Π is given by the induction hypothesis and

Δ is given by Lemma 6.2.

$$\begin{array}{c} \Pi \Big\| \\ [R, Q] \\ \Delta \Big\| \\ [R, (k, [\underbrace{c, \ldots, c}_{\times S(a_1)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_2)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_3)}]), (Q, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])] \end{array}$$

□

6.1.2. Completeness of the Encoding.

THEOREM 6.4. *For A, m, B, and S satisfying the constraints of 3-Partition, if there is a proof of $\theta(\langle A, m, B, S\rangle)$ in* FBV*, then the 3-Partition problem $\langle A, m, B, S\rangle$ is solvable.*

PROOF. By induction on m: The case for $m = 0$ corresponds to empty problem which is trivially solved. For the inductive case, let $\langle A, m+1, B, S\rangle$ be such that $A = \{a_1, a_2, \ldots, a_{3m}, a_{3m+1}, a_{3m+2}, a_{3m+3}\}$. Assuming that we have a proof of $\theta(\langle A, m+1, B, S\rangle)$, we show that $\langle A, m+1, B, S\rangle$ is solvable. Let

$$R = [(k, [\underbrace{c, \ldots, c}_{\times S(a_1)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_2)}]), \ldots, (k, [\underbrace{c, \ldots, c}_{\times S(a_{3m+2})}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_{3m+3})}])]$$

$$\text{and} \qquad Q = \underbrace{([\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})], \ldots, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])}_{\times m}$$

such that

$$\theta(\langle A, m+1, B, S\rangle) = [R, (Q, [\bar{k}, \bar{k}, \bar{k}, (\underbrace{\bar{c}, \ldots, \bar{c}}_{\times B})])] \quad .$$

From Corollary 5.34 we have that $\theta(\langle A, m+1, B, S\rangle)$ has a proof in FBV if and only if it has a proof in FBVs. It follows from Corollary 5.28 that

$$\begin{array}{c} [K_1, K_2] \\ \Delta \Big\|_{\mathsf{FBVs}} \\ R \end{array} \quad \text{such that} \quad \begin{array}{c} \Pi \Big\|_{\mathsf{FBVs}} \\ [K_1, Q] \end{array} \quad \text{and} \quad \begin{array}{c} \Big\|_{\mathsf{FBVs}} \\ [K_2, \bar{k}, \bar{k}, \bar{k}, (\bar{c}, \ldots, \bar{c})] \end{array} \quad .$$

Because there are only positive atoms in R, it follows that none of the rules $\mathsf{ai}{\downarrow}$ and is can be applied in Δ, hence the derivation Δ must be the structure R. This implies that $[K_1, K_2]$ are two partitions of R. Observe that in K_2 there must be exactly 3 occurrences of k, which implies that, for some $a_i, a_j, a_k \in A$,

$$K_2 = [(k, [\underbrace{c, \ldots, c}_{\times S(a_i)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_j)}]), (k, [\underbrace{c, \ldots, c}_{\times S(a_k)}])]$$

and $S(a_i) + S(a_j) + S(a_k) = B$, and we can apply the induction hypothesis to the proof Π. □

COROLLARY 6.5. *System* FBV *is NP-hard.*

PROOF. Follows immediately from Theorem 6.3 and Theorem 6.4. □

Because system BV is a conservative extension of system FBV, this result implies the NP-hardness of system BV.

COROLLARY 6.6. *System* BV *is NP-hard.*

PROOF. Follows immediately from Proposition 2.30 and Corollary 6.5. □

6.2. System BV is NP-complete

With Proposition 5.14, we have seen that the length of a proof of a BV structure is bounded by a polynomial in the size of this structure. Thus, the main result of this Chapter follows from the result in Sections 6.1.

THEOREM 6.7. *System* BV *is NP-complete.*

PROOF. Follows immediately from Corollary 6.6 and Proposition 5.14. □

COROLLARY 6.8. *Multiplicative linear logic extended by the rules* mix *and* mix0*, or System* FBV*, is NP-complete.*

PROOF. Follows immediately from Corollary 6.5 and Proposition 5.14. □

CHAPTER 7

Implementing Deep Inference Imperatively

In the previous chapters, we have seen implementations of the calculus of structures systems in Maude. Due to its simple high level language and built in breadth-first function, Maude is well suited for implementing these systems. However, when proof search is considered by using a search strategy different than breadth-first search, implementing these strategies in Maude is rather intricate due to the interweaving between the object-level language and the complex meta-level language of Maude. As an alternative to these Maude implementations, in this chapter, I present a recipe for implementing the systems of the calculus of structures in imperative languages, such as C and Java. In these languages different search strategies can be easily implemented and advanced programming techniques can be effectively used.

In the following, I will describe a Java implementation of system BV. Because imperative languages usually do not support pattern matching and term rewriting directly, in these implementations the Tom tool is used. Tom [**MRV03, KMR05b**] is a pattern matching preprocessor that integrates term rewriting and pattern matching facilities into imperative and functional languages such as C, Java and OCaml. By resorting to these features of Tom, it becomes possible to combine term rewriting with the expressive power of these languages.

The Tom tool does not support associative commutative term rewriting. For this reason, instead of expressing commutativity as equations in the underlying equational theory of a calculus of structures system, I show that the role played by the equations for commutativity can be embedded into the inference rules of the system. Given that the equations for commutativity are equivalently removed, associativity of the structures can be expressed in a list representation of the structures. Then, by expressing the inference rules as term rewriting rules as before, these systems can be easily implemented.

7.1. Removing the Equations for Commutativity

In this section, I will present systems in the calculus of structures where the equations for commutativity become redundant, and thus can be equivalently removed from the underlying equational theory. In order to remove the equations for commutativity, I will make the role played by these equations explicit in the inference rules. That is, for every possible instance of the inference rules which is obtained by the applications of the equations for commutativity, I will introduce an inference rule that simulates the role played by these equations. I will first consider system BV and then show that other systems in the calculus of structures can be treated analogously, e.g., system KSg:

7.1.1. Removing the Equations for Commutativity in System BV.

DEFINITION 7.1. *Consider the following restriction on system* BVu*, given in Figure 4.4: The structures W in the inference rules are restricted to atoms, copar structures and seq structures. In other words, structure W is not a proper par structure. We will call this system* unit-free lazy BV *or* BVul.

PROPOSITION 7.2. *System* BV *and system* BVul *are equivalent.*

PROOF. Follows immediately from Corollary 4.33 and Proposition 5.39. □

DEFINITION 7.3. *The system in Figure 7.1 is called* commutativity-free BV *or* BVc*, where W is not a proper par structure. Inference rules of system* BVc *are applied to* BV *structures that are considered equivalent only modulo equations for associativity.*

PROPOSITION 7.4. *System* BV *and system* BVc *are equivalent.*

PROOF. Inference rules of BVc are instances of the inference rules of BV. The proof of the other direction is by induction on the length of the proof with case analysis on the last applied rule: Let Π be the proof of R in BVul. By induction on Π, we construct a proof Π' of R in BVc.

- If Π is $ax\,\dfrac{}{[a,\bar{a}]}$, take the same rule in BVc. (Observe that $ax\,\dfrac{}{[\bar{a},a]}$ is an instance of this rule, also when commutativity does not apply, since $\bar{a}$ is an atom, and $\bar{\bar{a}} = a$.)

- If $\mathsf{ai}_1{\downarrow}$ is the last rule applied in Π, we have the following 4 cases:

  (1) $\overset{\mathsf{ai}_1{\downarrow}}{\approx}\dfrac{S\{R\}}{\dfrac{S[R,[a,\bar{a}]]}{Q}}$, there are the following possibilities for Q : If
  - $Q = S[R,a,\bar{a}]$; take $\mathsf{ai}_{11}{\downarrow}$.
  - $Q = S[a,\bar{a},R]$; take $\mathsf{ai}_{12}{\downarrow}$.
  - $Q = S[a,R,\bar{a}]$; take $\mathsf{ai}_{13}{\downarrow}$.

  (2) $\overset{\mathsf{ai}_1{\downarrow}}{\approx}\dfrac{S\{R\}}{\dfrac{S(R,[a,\bar{a}])}{Q}}$, there are the following possibilities for Q : If
  - $Q = S(R,[a,\bar{a}])$; take $\mathsf{ai}_{21}{\downarrow}$.
  - $Q = S([a,\bar{a}],R)$; take $\mathsf{ai}_{22}{\downarrow}$.

  (3) $\mathsf{ai}_1{\downarrow}\,\dfrac{S\{R\}}{S\langle R;[a,\bar{a}]\rangle}$, then take $\mathsf{ai}_3{\downarrow}$.

  (4) $\mathsf{ai}_1{\downarrow}\,\dfrac{S\{R\}}{S\langle [a,\bar{a}];R\rangle}$, then take $\mathsf{ai}_4{\downarrow}$.

- If s_1 is the last rule applied in Π, we have the following 4 cases

$$ax\,\frac{}{[a,\bar{a}]} \qquad \mathsf{ai}_{11}{\downarrow}\,\frac{S\{R\}}{S[R,a,\bar{a}]} \qquad \mathsf{ai}_{12}{\downarrow}\,\frac{S\{R\}}{S[a,\bar{a},R]} \qquad \mathsf{ai}_{13}{\downarrow}\,\frac{S\{R\}}{S[a,R,\bar{a}]}$$

$$\mathsf{ai}_{21}{\downarrow}\,\frac{S\{R\}}{S(R,[a,\bar{a}])} \qquad \mathsf{ai}_{22}{\downarrow}\,\frac{S\{R\}}{S([a,\bar{a}],R)} \qquad \mathsf{ai}_{3}{\downarrow}\,\frac{S\{R\}}{S\langle R;[a,\bar{a}]\rangle} \qquad \mathsf{ai}_{4}{\downarrow}\,\frac{S\{R\}}{S\langle[a,\bar{a}];R\rangle}$$

$$\mathsf{s}_{11a}\,\frac{S([R,W],T)}{S[(R,T),W]} \qquad \mathsf{s}_{12a}\,\frac{S([R,W],T)}{S[(T,R),W]} \qquad \mathsf{s}_{13a}\,\frac{S([R,W],T)}{S[W,(R,T)]} \qquad \mathsf{s}_{14a}\,\frac{S([R,W],T)}{S[W,(T,R)]}$$

$$\mathsf{s}_{15a}\,\frac{S([(R,U),W],T)}{S[(R,T,U),W]} \qquad \mathsf{s}_{16a}\,\frac{S([(R,U),W],T)}{S[W,(R,T,U)]}$$

$$\mathsf{s}_{11b}\,\frac{S[([R,W],T),P]}{S[(R,T),P,W]} \qquad \mathsf{s}_{12b}\,\frac{S[([R,W],T),P]}{S[(T,R),P,W]}$$

$$\mathsf{s}_{13b}\,\frac{S[([R,W],T),P]}{S[W,P,(R,T)]} \qquad \mathsf{s}_{14b}\,\frac{S[([R,W],T),P]}{S[W,P,(T,R)]}$$

$$\mathsf{s}_{15b}\,\frac{S[([[R,U),W],T),P]}{S[(R,T,U),P,W]} \qquad \mathsf{s}_{16b}\,\frac{S[([[R,U),W],T),P]}{S[W,P,(R,T,U)]}$$

$$\mathsf{q}_{11}{\downarrow}\,\frac{S\langle[R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]} \qquad \mathsf{q}_{12}{\downarrow}\,\frac{S[\langle[R,T];[U,V]\rangle,P]}{S[\langle R;U\rangle,P,\langle T;V\rangle]}$$

$$\mathsf{q}_{21}{\downarrow}\,\frac{S\langle R;T\rangle}{S[R,T]} \qquad \mathsf{q}_{22}{\downarrow}\,\frac{S\langle R;T\rangle}{S[T,R]} \qquad \mathsf{q}_{23}{\downarrow}\,\frac{S[\langle R;T\rangle,P]}{S[R,P,T]} \qquad \mathsf{q}_{24}{\downarrow}\,\frac{S[\langle R;T\rangle,P]}{S[T,P,R]}$$

$$\mathsf{q}_{31}{\downarrow}\,\frac{S\langle[W,T];U\rangle}{S[W,\langle T;U\rangle]} \qquad \mathsf{q}_{32}{\downarrow}\,\frac{S\langle[W,T];U\rangle}{S[\langle T;U\rangle,W]} \qquad \mathsf{q}_{33}{\downarrow}\,\frac{S[\langle[W,T];U\rangle,P]}{S[W,P,\langle T;U\rangle]}$$

$$\mathsf{q}_{34}{\downarrow}\,\frac{S[\langle[W,T];U\rangle,P]}{S[\langle T;U\rangle,P,W]} \qquad \mathsf{q}_{44}{\downarrow}\,\frac{S[\langle T;[W,U]\rangle,P]}{S[\langle T;U\rangle,P,W]}$$

$$\mathsf{q}_{41}{\downarrow}\,\frac{S\langle T;[W,U]\rangle}{S[W,\langle T;U\rangle]} \qquad \mathsf{q}_{42}{\downarrow}\,\frac{S\langle T;[W,U]\rangle}{S[\langle T;U\rangle,W]} \qquad \mathsf{q}_{43}{\downarrow}\,\frac{S[\langle T;[W,U]\rangle,P]}{S[W,P,\langle T;U\rangle]}$$

FIGURE 7.1. System BVc

(1) $\overset{\mathsf{s}_1}{\approx}\,\dfrac{\dfrac{S([R,W],T)}{S[(R,T),W]}}{Q}$, there are the following possibilities for Q : If

- $Q = S[(R,T),W]$; take s_{11a}.
- $Q = S[(T,R),W]$; take s_{12a}.
- $Q = S[W,(R,T)]$; take s_{13a}.
- $Q = S[W,(T,R)]$; take s_{14a}.

(2) $\overset{\mathsf{s}_1}{\approx} \dfrac{\dfrac{S([(R,U),W],T)}{S[(R,U,T),W]}}{Q}$, there are the following possibilities for Q : If

- $Q = S[(R,T,U),W]$; take s_{15a}.
- $Q = S[W,(R,T,U)]$; take s_{16a}.

(3) $\overset{\mathsf{s}_1}{\approx} \dfrac{\dfrac{S([R,W],T)}{S[(R,T),W]}}{Q}$, there are the following possibilities for Q : If

- $Q = S'[(R,T),P,W]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{11b}.
- $Q = S'[(T,R),P,W]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{12b}.
- $Q = S'[W,P,(R,T)]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{13b}.
- $Q = S'[W,P,(T,R)]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{14b}.

(4) $\overset{\mathsf{s}_1}{\approx} \dfrac{\dfrac{S([R,W],T)}{S[(R,T),W]}}{Q}$, there are the following possibilities for Q : If

- $Q = S'[(R,T,U),P,W]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{15b}.
- $Q = S'[W,P,(R,T,U)]$ and $S\{\ \} = S'[\{\ \},P]$; take s_{16b}.

- If $\mathsf{q}_1{\downarrow}$ is the last rule applied in Π, such that

$\overset{\mathsf{q}_1{\downarrow}}{\approx} \dfrac{\dfrac{S\langle[R,T];[U,V]\rangle}{S[\langle R;U\rangle,\langle T;V\rangle]}}{Q}$, there are the following possibilities for Q : If

- $Q = S[\langle R;U\rangle,\langle T;V\rangle]$; take $\mathsf{q}_{11}{\downarrow}$.
- $Q = S'[\langle R;U\rangle,P,\langle T;V\rangle]$ and $S\{\ \} = S'[\{\ \},P]$; take $\mathsf{q}_{12}{\downarrow}$.

- If $\mathsf{q}_2{\downarrow}$ is the last rule applied in Π, such that

$\overset{\mathsf{q}_2{\downarrow}}{\approx} \dfrac{\dfrac{S\langle R;T\rangle}{S[R,T]}}{Q}$, there are the following possibilities for Q : If

- $Q = S[R,T]$; take $\mathsf{q}_{21}{\downarrow}$.
- $Q = S[T,R]$; take $\mathsf{q}_{22}{\downarrow}$.

– $Q = S'[R, P, T]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{23}\downarrow$.
– $Q = S'[T, P, R]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{24}\downarrow$.

- If $\mathsf{q}_3\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_3\downarrow \frac{S\langle [W, T]; U\rangle}{S[W, \langle T; U\rangle]} \approx \frac{}{Q}$$

, there are the following possibilities for Q : If

– $Q = S[W, \langle T; U\rangle]$; take $\mathsf{q}_{31}\downarrow$.
– $Q = S[\langle T; U\rangle, W]$; take $\mathsf{q}_{32}\downarrow$.
– $Q = S'[W, P, \langle T; U\rangle]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{33}\downarrow$.
– $Q = S'[\langle T; U\rangle, P, W]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{34}\downarrow$.

- If $\mathsf{q}_4\downarrow$ is the last rule applied in Π, such that

$$\mathsf{q}_4\downarrow \frac{S\langle T; [W, U]\rangle}{S[W, \langle T; U\rangle]} \approx \frac{}{Q}$$

, there are the following possibilities for Q : If

– $Q = S[W, \langle T; U\rangle]$; take $\mathsf{q}_{41}\downarrow$.
– $Q = S[\langle T; U\rangle, W]$; take $\mathsf{q}_{42}\downarrow$.
– $Q = S'[W, P, \langle T; U\rangle]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{43}\downarrow$.
– $Q = S'[\langle T; U\rangle, P, W]$ and $S\{\ \} = S'[\{\ \}, P]$; take $\mathsf{q}_{44}\downarrow$.

□

7.1.2. Removing the Equations for Commutativity in System KSg**.** We can carry the above ideas analogously to other systems of the calculus of structures, e.g., system KSg:

Definition 7.5. *The system in Figure 7.2 is called* commutativity-free KSg *or* KSgc*, where W is either an atom or a conjunction. Inference rules of system* KSgc *are applied to* KSg *structures that are considered equivalent modulo equations for associativity.*

Proposition 7.6. *System* KSg *and system* KSgc *are strongly equivalent.*

Proof. It is immediate that the inference rules of KSgc are instances of the inference rules of KSg. For the proof of the other direction observe that from Proposition 4.53 systems KSg and KSgn are strongly equivalent. By inductive case analysis on the commutative application of the inference rules of KSgn construct a derivation in KSgc: Observe that the switch rule can be replaced with its lazy version in system KSgn analogous to that of system BVul. The case for the switch rule being same as that of Proposition 7.4 other cases follow trivially by analogous case analysis. □

$$\mathsf{t}\!\downarrow \frac{}{\mathsf{t}} \qquad \mathsf{ai}_{11}\!\downarrow \frac{S\{\mathsf{t}\}}{S[a,\bar{a}]} \qquad \mathsf{ai}_{12}\!\downarrow \frac{S[R,\mathsf{t}]}{S[a,R,\bar{a}]} \qquad \mathsf{w}\!\downarrow \frac{S\{\mathsf{f}\}}{S\{R\}} \qquad \mathsf{c}\!\downarrow \frac{S[R,R]}{S\{R\}}$$

$$\mathsf{u}_{1l}\!\downarrow \frac{S\{R\}}{S[\mathsf{f},R]} \qquad \mathsf{u}_{1r}\!\downarrow \frac{S\{R\}}{S[R,\mathsf{f}]} \qquad \mathsf{u}_{2l}\!\downarrow \frac{S\{R\}}{S(\mathsf{t},R)} \qquad \mathsf{u}_{2r}\!\downarrow \frac{S\{R\}}{S(R,\mathsf{t})}$$

$$\mathsf{s}_{11a}\,\frac{S([R,W],T)}{S[(R,T),W]} \qquad \mathsf{s}_{12a}\,\frac{S([R,W],T)}{S[(T,R),W]} \qquad \mathsf{s}_{13a}\,\frac{S([R,W],T)}{S[W,(R,T)]} \qquad \mathsf{s}_{14a}\,\frac{S([R,W],T)}{S[W,(T,R)]}$$

$$\mathsf{s}_{15a}\,\frac{S([(R,U),W],T)}{S[(R,T,U),W]} \qquad \mathsf{s}_{16a}\,\frac{S([(R,U),W],T)}{S[W,(R,T,U)]}$$

$$\mathsf{s}_{11b}\,\frac{S[([R,W],T),P]}{S[(R,T),P,W]} \qquad \mathsf{s}_{12b}\,\frac{S[([R,W],T),P]}{S[(T,R),P,W]}$$

$$\mathsf{s}_{13b}\,\frac{S[([R,W],T),P]}{S[W,P,(R,T)]} \qquad \mathsf{s}_{14b}\,\frac{S[([R,W],T),P]}{S[W,P,(T,R)]}$$

$$\mathsf{s}_{15b}\,\frac{S[([(R,U),W],T),P]}{S[(R,T,U),P,W]} \qquad \mathsf{s}_{16b}\,\frac{S[([(R,U),W],T),P]}{S[W,P,(R,T,U)]}$$

FIGURE 7.2. System KSgc

7.1.3. Nondeterminism in System BVc. With the definition below, I will combine the ideas from systems BVc and BVi in a single system, that is, I will impose the restrictions on the rules of BVi analogously on the inference rules of system BVc. This way, a system will be obtained where the equalities for unit and commutativity are redundant and nondeterminism is reduced.

DEFINITION 7.7. *Let* commutativity-free interaction system BV *or system* BVci *be the system obtained by imposing the following restrictions on system* BVc*: In the rules* $\mathsf{s}_{11a},\mathsf{s}_{12a},\mathsf{s}_{13a},\mathsf{s}_{14a},\mathsf{s}_{11b},\mathsf{s}_{12b},\mathsf{s}_{13b}$*, and* s_{14b} *we have* $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,W \neq \emptyset$*; in the rules* $\mathsf{s}_{15a},\mathsf{s}_{16a},\mathsf{s}_{15b}$*, and* s_{16b} *we have* $\mathsf{at}\,\overline{(R,U)} \cap \mathsf{at}\,W \neq \emptyset$*; in the rules* $\mathsf{q}_{11}\!\downarrow$*, and* $\mathsf{q}_{12}\!\downarrow$ *we have* $\mathsf{at}\,\overline{R} \cap \mathsf{at}\,T \neq \emptyset$ *and* $\mathsf{at}\,\overline{U} \cap \mathsf{at}\,V \neq \emptyset$*; in the rules* $\mathsf{q}_{31}\!\downarrow,\mathsf{q}_{32}\!\downarrow,\mathsf{q}_{33}\!\downarrow$*, and* $\mathsf{q}_{34}\!\downarrow$ *we have* $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,T \neq \emptyset$*; in the rules* $\mathsf{q}_{41}\!\downarrow,\mathsf{q}_{42}\!\downarrow,\mathsf{q}_{43}\!\downarrow$*, and* $\mathsf{q}_{44}\!\downarrow$ *we have* $\mathsf{at}\,\overline{W} \cap \mathsf{at}\,U \neq \emptyset$*.*

PROPOSITION 7.8. *If systems* BV *and* BVi *are equivalent then system* BV *and system* BVci *are equivalent.*

PROOF. Follows immediately from Proposition 7.4. □

7.2. Implementation of BV in Java

In the previous sections, we have seen that it is possible to remove the equations up to associativity in a calculus of structures system. In the following by resorting to these results and using a list representation of n-ary terms, which captures associativity, I will present a Java implementation of the term rewriting system corresponding to system BVci by using the term rewriting features provided by Tom.

Tom is a language extension that adds pattern-matching facilities to existing languages like Java, C, and OCaml. This approach is particularly well-suited when describing transformations of structured expressions like trees/terms. Exploiting these features, I use Tom, combined with Java, to implement proof search in system BV.

Design and implementation issues related to Tom can be found in [**MRV03, KMR05b**]. However, let me briefly describe this tool, following [**KMR05b**]: At a level of abstraction, we can say that Tom adds two new constructs to the imperative language: `%match` and *back-quote* (‘). The first construct is similar to the `match` primitive of ML and related languages: Given a term (called subject) and a list of pairs pattern-action, the `match` primitive selects a pattern that matches the subject and performs the associated action.

A main originality of this system is to be data-structure independent. This means that a *mapping* has to be defined to connect algebraic data-structures to low-level data-structures that correspond to the implementation. In such a setting, pattern matching is performed on the algebraic data-structures. Most of the time, Tom is used in conjunction with the ApiGen system [**vdBMV03**], which generates abstract syntax tree implementations and a mapping from a given datatype definition. The input format for ApiGen is a concise language defining sorts and constructors for the abstract syntax. The output is an efficient, in time and memory, (Java) implementation for this datatype. This implementation is characterized by strong typing and maximal sub-term sharing, while providing both memory efficiency and constant-time equality checking.

7.2.1. Structures as Data Structures. A difficulty when implementing the systems of the calculus of structures is to find an appropriate representation for the structures. Below, these constructors are considered as unary operators that take a *list of structures* as argument. Using ApiGen, the considered data-type can be described by the following signature, demonstrated for the structures *par*, *cop*, and *seq* of system BV:

```
module Struct
  public sorts Struc StrucPar StrucCop StrucSeq
  abstract syntax
    a -> Struc
    b -> Struc
    ...other atom constants
    neg(Struc)        -> Struc
    par(StrucPar)     -> Struc
    cop(StrucCop)     -> Struc
    seq(StrucSeq)     -> Struc
    concPar( Struc* ) -> StrucPar
```

```
concCop( Struc* ) -> StrucCop
concSeq( Struc* ) -> StrucSeq
```

The grammar rule `par(StrucPar) -> Struc` defines a unary operator `par` of sort `Struct` that takes a `StrucPar` as unique argument. The grammar rule `concPar(Struc*) -> StrucPar` defines the `concPar` operator of sort `StrucPar`. The special syntax `Struc*` indicates that `concPar` is a *list-operator* that takes a list of `Struc` as argument. Thus, by combining `par` and `concPar` it becomes possible to represent the structure $[a,[b,c]]$ by `par(concPar(a,b,c))`. Note that structures are flattened meaning that unnecessary brackets are removed. In Tom, list-operators are convenient because their arity is not fixed. Thus, `concPar(a,b,c)` corresponds to a list of 3 elements, `concPar(a)` corresponds to a list of single element, namely `a`, whereas `concPar()` denotes the empty list. (R,T) and $\langle R;T\rangle$ are represented in a similar way, using `cop, seq`, `concCop`, and `concSeq`.

A problem with this approach is that objects such as `par(concPar())` can be manipulated, although such objects do not necessarily correspond to structures that we intend to manipulate. It is also possible to have several representations for the same structure. Hence, `par(concPar(a))` and `cop(concCop(a))` both denote the structure `a`. To avoid such situations, in the defined mapping a notion of canonical form is encoded. This avoids building such unintended terms.

- $[]$, $\langle\rangle$ and $()$ are reduced when containing only one sub-structure: $par(concPar(x)) \to x$
- nested structures are flattened: $par(concPar(...,par(concPar(x_1,...,x_n)),...) \to par(concPar(...,x_1,...,x_n,...)$
- subterms are sorted (according to a given total lexical order $<$): $concPar(...,x_i,...,x_j,...) \to concPar(...,x_j,...,x_i,...)$ if $x_j < x_i$.

This notion of canonical form allows to efficiently check if two terms represent the same structure with respect to commutativity of those logical operators.

7.2.2. Rewrite rules. The rewrite rules define the deduction steps in system BVci. They are implemented by a *match* construct that matches a sub-term with the left-hand side of the rewrite rule. Then the right-hand side of the rule builds the deduced structure.

For instance, the rules $[(R,T),U] \to ([R,U],T)$ and $[(R,T),U] \to ([T,U],R)$ are encoded by the following match construct.

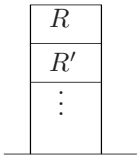
```
%match(Struc t) {
  par(concPar(X1*,cop(concCop(R*,T*)),X2*,U,X3*)) -> {
    if(‘T*.isEmpty() || ‘R*.isEmpty() ) {
    } else {
      StrucPar context = ‘concPar(X1*,X2*,X3*);
      if(canReact(‘R*,‘U)) {
        StrucPar parR = cop2par(‘R*);
                          // transform a StrucCop into a StrucPar
        Struc elt1 = ‘par(concPar(
                 cop(concCop(par(concPar(parR*,U)),T*)),context*));
        c.add(elt1);
      }
      if(canReact(‘T*,‘U)) {
        StrucPar parT = cop2par(‘T*);
```

```
        Struc elt2 = `par(concPar(
                cop(concCop(par(concPar(parT*,U)),R*)),context*));
        c.add(elt2);
}  }  }  }
```

The first test in the above code ensures that the rules are not applied if `R` or `U` is the empty list. The other tests implement the restrictions on the application of the rules given in Chapter 5 for reducing nondeterminism. This is done by using an auxiliary predicate function `canReact(R,U)` that collects all atoms in the structures `R` and `U` and returns `true` only if `R` contains at least one atom that is contained in a negated form in `U`. This function can be made efficient by using the features of the host language of Tom, in the case of the present implementation, by using an efficient hash-set implementation in Java. The remaining rules are expressed in a similar way.

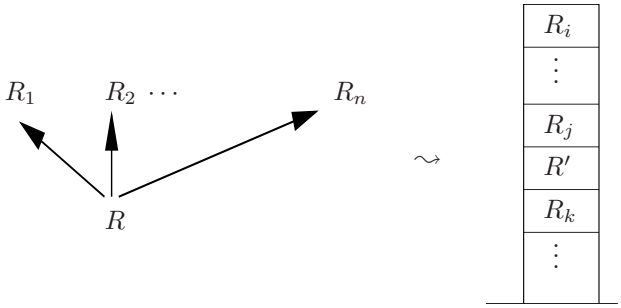
7.2.3. Strategy. When designing a proof search procedure, implementing the set of inference rules is very important, but this is only one part of the job. The second part consists in defining a strategy that describes how to apply the rules. In rule based systems like ELAN or Maude, such strategies can be described by using primitive operators or meta-level capabilities. In some cases, however, it may be difficult to express strategies that take time and space into consideration. In ELAN for example, the search is implemented using a backtracking mechanism. This is a good approach to implement depth-first search strategies. While being efficient in space, such a strategy may lead to explore infinite branches and non-terminating programs. On the other hand, breadth-first search, as in Maude, *eventually* terminates when a proof exists, but the memory needed can be huge. Further, given that every structure has at least several child nodes after applying all the possible rule instances, the search space explodes after few steps and this results in infeasible amount of time for the search to terminate. In Tom, there is no particular support for implementing search space exploration strategies. Thus, the search space has to be handled explicitly. On one hand, this leads to more complex implementations, but on the other, this allows to define different search strategies that involve heuristic functions, or implement randomized search algorithms, e.g., hill climbing [**RN02**].

In the implementation that I discuss here, a global search strategy has been employed. However, by using the same ideas and by modifying the implementation slightly, other search strategies can be easily implemented. Let me describe how this is done: The search space is given by a stack of structures. At the beginning of the search the stack consists of a single structure, namely, the structure the proof of which is being searched.



The algorithm (see, e.g., [**RN02**]) takes the top most structure R in the stack and applies to R all the possible bottom-up rule instances, premises of which are R_1, R_2,..., R_n. Then all these structures are placed into the stack with respect to a heuristic function f such that there is a total order of structures in the stack,

e.g., $f(R_i) > \ldots > f(R_j) > f(R') > f(R_k)$.



This is repeated until the top-most structure in the stack is the unit.

Global search strategy can be easily modified to local search by putting a fixed bound on the size of the stack. Further, breadth-first search can be introduced by stacking the structures R_1, R_2,..., R_n at the bottom of the stack; depth-first search can be introduced by stacking R_1, R_2,..., R_n at the top of the stack.

In the implementation of system BVci, which I discuss here, in a global search setting, the following heuristic function f on structure R is employed:

$$f(R) = \frac{1}{(\#\text{ of ';' in } R)\,.\,(\#\text{ of atoms in } R)^2}$$

This heuristic function delays the instances of the rule $\mathsf{q}_2{\downarrow}$, because the structures with less number of ';' symbols are pushed to the top of the stack. It promotes the instances of the rule $\mathsf{ai}{\downarrow}$, because the structures with less number of atoms are also pushed to the top of the stack by this function. Table 7.1 gives a performance comparison of the implementations of system BVi in Maude and system BVci in Tom on the examples given Subsection 4.3.1.

7.3. Discussion

In this chapter, we have seen a recipe for implementing systems of the calculus of structures in imperative languages such as C and Java where these languages can be used in their full expressive power. As an example to such an implementation, we have seen a proof search implementation of system BV in Java. In this implementation the pattern matching preprocessor Tom is used in order to express the inference rules as term rewriting rules within a Java program. The source code of the implementation is available at the Tom distribution [1]. A representative applet of this implementation is also available online [2].

In the previous chapters, we have seen that the systems of the calculus of structures can be expressed as term rewriting systems modulo equational theories and this equational theory can be reduced to equations for associativity and commutativity. Because Tom does not support associative-commutative rewriting, in this chapter, by making the role played by the equations for commutativity in the application of the inference rules explicit, I introduced a system equivalent to system BV where these equations become redundant. This makes it possible to express the associativity of structures in a list representation.

[1] `http://tom.loria.fr`

[2] `http://tom.loria.fr/examples/structures/BV.html`

Query	System	finds a proof in # millisec.
1.	Maude	60
	Tom	300
2.	Maude	20
	Tom	16
3.	Maude	120
	Tom	29
4.	Maude	140
	Tom	26
5.	Maude	160
	Tom	15
6.	Maude	550
	Tom	694
7.	Maude	900
	Tom	2172
8.	Maude	1270
	Tom	14442
9.	Maude	2830
	TomBVi	1553
10.	Maude	3670
	Tom	out of memory

Table 7.1. Representative performance comparison of proof search in the implementations of system BVi in Maude and system BVci in Tom.

We have also seen that the procedure for removing the equations for commutativity can be analogously generalized to other systems in the calculus of structures. As an evidence for this, I have introduced a system for classical logic that is equivalent to system KSg where the equations for commutativity are redundant.

Because of the expressive power of imperative languages, by following the recipe described in this chapter, it becomes possible to easily implement any search strategy for proof search. In the implementation described in this chapter a *global search* strategy is employed: Stack the structures that are premises of all the bottom-up instances of the inference rules with respect to a heuristic function and proceed with applying this procedure to the topmost structure in the stack until the topmost structure is the unit. This allows to choose a heuristic function which respects the mutual relations between dual atoms such that proofs can be constructed by annihilating dual atoms.

Because systems in the calculus of structures follow a common scheme which I exploit in this chapter, the content of this chapter can be analogously carried over to any other system in the calculus of structures. Thus, the description of the implementation that I describe provides a recipe for implementing systems for other logics in the calculus of structures. Further, the implementation described in this chapter can be easily generalized for implementing different tools for the other systems of the calculus of structures, also by employing different search strategies at will.

CHAPTER 8

A Common Language for Planning and Concurrency

Planning and concurrency are two fields of computer science and AI that evolved independently. However, as we have seen in Section 1.3, although these two fields address problems which are different in perspective, they aim at solving tasks that are very similar in nature. The difference in perspective can be seen as the difference of quantification: Planning formalisms focus on finding a plan, if there *exists* such a plan, that solves a given planning problem. The focus in concurrency theory is on the global behavior of a given concurrent system, resulting in *universally quantified* queries such as verification of a security protocol. In contrast to the approaches to planning, in order to be able to handle such queries, languages for concurrency are equipped with a rich arsenal of mathematical methods that allow for an analysis of equivalence of processes.

In concurrency theory, the interaction between the processes of a concurrent system is central: The input produced by one process is the output of another process which is consumed during the interaction of these two processes. That is, during their interaction, the input of the latter annihilates (consumes) the output of the former, and this way the latter process produces its output. This output is then to be consumed by another process, and so on. This scheme of causality is also captured in resource conscious planning (see Section 8.2). Further, in a possible model of concurrent processes, when two processes do not require the same resources as input, they can co-occur. The interaction of such processes with their common descendants and succedents synchronizes these processes. Their independence with respect to the resources that they require as input gives an explicit representation of nondeterminism. Such a scheme of causality, independence and nondeterminism by means of resources is captured, for instance, in Petri nets [**Pet62**]. As in petri nets, by observing these interactions, due to the representation of resources being consumed and produced, conclusions about the global behavior of the system can be drawn.

In this chapter, I present a common proof theoretical language for planning and concurrency. This language, which I call $\mathcal{K}$, aims at bringing planning and concurrency closer, so that the tools and methods used in these fields can be carried over both ways. Thus, the goal of this language is, while remaining in formal grounds, to act as a bridge between these two fields so that techniques can be interchanged: Such a language is useful for bringing the formal methods of concurrency theory to planning. By means of this language, one can address questions for plans that are standard in concurrency theory, for instance, in order to establish a notion of plan equivalence analogous to the notion of equivalence of processes.

Such a language also prepares the formal grounds for bringing highly optimized implementation techniques from planning to concurrency and vice versa.

Perhaps the most important question in designing a common logical language for planning and concurrency is which logic to choose. The host logic should be expressive enough to capture causality so that one can do planning and simultaneously provide a satisfactory semantic treatment of concurrent actions. From the planning point of view, the underlying logic must be powerful enough to express causality in a simple way without raising the frame problem (see Subsection 8.1.6). From the concurrency point of view, an explicit treatment of resources is crucial in order to express the independence and nondeterminism in a concurrent system. Further, in process algebras, which study the syntactic representations of processes, parallel and sequential composition are represented at the same level because they are equivalently important notions for expressing concurrent processes. Thus, it is crucial to express parallel and sequential composition of actions at the same logical level. This way, the structure of the problem can be captured at the logical level, rather than term level, so that logic can be used in an interesting and useful way to do reasoning on these expressions.

The linear logic approach to planning (see Section 8.2) offers a solution to some of these challenges. Although parallel composition of actions can be naturally mapped to the commutative *par* operator of linear logic, sequential composition does not find a natural interpretation. For this reason, for the language I develop, I resort to system NEL. System NEL provides a satisfactory treatment of resources and allows to represent the parallel and sequential composition by means of its logical operators. Parallel composition of actions, plans, and processes is naturally mapped to the commutative *par* operator of linear logic, whereas sequential composition is mapped to the non-commutative self-dual *seq* operator.

From the planning point of view, the language $\mathcal{K}$ follows the linear logic approach to conjunctive planning [**MTV90**], which is, in [**GHS96**], shown to be equivalent to Bibel's connection method approach [**Bib86**] and Hölldobler and Schneeberger's equational logic programming approach [**HS90**]. From the concurrency point of view, the language has the features of a simple process algebra corresponding to a fragment of CCS [**Mil89**] equipped with prefixing (sequential composition) and parallel composition. The language admits a behavioral noninterleaving branching time concurrency semantics, namely labelled event structure semantics [**SNW96, WN95**]. As other approaches to conjunctive planning, the language resembles Petri nets. The computation in language $\mathcal{K}$ is performed as computation as proof search in an abstract logic programming setting.

In the following section, I will give an overview of the previous work on reasoning about action in artificial intelligence and planning. This section can be read as a survey.

8.1. Planning: Historical Perspective

Reasoning about action and planning are the fields of computer science that study the characterization of the concepts of action, change, and planning of action sequences to accomplish a given task. Mainly, logic has been providing the rigorous methods that are necessary for the formal treatment of the problems addressed in these fields. Planning systems, as algorithms that operate on explicit representations of states and actions, have been historically motivated by theorem proving,

state space search, and associated techniques and the needs of robotics [**RN02**]. The relation between logic, changes involved in reasoning, and plan generation have been studied, either by embedding actions into a classical logic framework or by using non-standard formalisms.

Situation calculus, which is a formalism for reasoning about action in classical logic, initiated the declarative (theorem proving) approach, in contrast to procedural approach: In the declarative approach, the agent's initial program, before it starts to receive percepts, is built by adding one by one the sentences that represent the designer's knowledge of the environment [1].

8.1.1. Situation Calculus and the Like. The research on reasoning about action has been mainly driven by the so called *frame problem* since it was recognized by McCarthy and Hayes in [**MH69**]. Informally, the frame problem occurs when the formal language expresses what changes, but does not express what stays the same. In other words, representing all the things that stay the same is called the frame problem. The name "frame" comes from "frame of reference" in physics, the assumed stationary background with respect to which motion is measured. It also has an analogy to the frames of a movie, in which normally very little changes from one frame to the next. Finding an efficient solution to the frame problem is important, because in the real world almost everything stays the same most of the time. Each action affects only a tiny fraction of the world. In this respect, the frame problem has long been recognized as a key problem within formal theories of action and has been studied by many authors.

Situation calculus as a formalism was first proposed in [**McC63**] and elaborated in [**MH69**]. However, the name "situation calculus" was first used in [**MH69**]. [**McC86**] is the first significant, but unsuccessful, attempt to solve the frame problem within the situation calculus. The version of the situation calculus that was developed for the cognitive robotics project at the University of Toronto is perhaps the melting pot of all the others with the same name. [**Sha97**] and [**Rei01**] give complete treatments of reasoning about action in situation calculus along these lines.

In the simplest form of situation calculus, each action, which is a term built from function symbols of a classical logic signature, is described by two axioms: A possibility axiom that states when it is possible to execute an action, and an effect axiom (successor-state axiom) that states what happens when an action is executed [**Rei91**]. Often $Poss(a, s)$ is used to express that it is possible to execute action a in situation s. Situations are logical terms built from a function symbol representing the initial situation and another function symbol representing execution of actions: The function $Result(a, s)$ (sometimes called Do) names the situation that results when action a is executed in situation s. Fluents are predicates that represent atomic properties of the world and vary from one situation to the next, e.g., the location of an agent. According to the dictionary, a fluent is something that flows, like a liquid. In this sense, it is flowing or changing across situations. The principal intuition captured by the axioms is that situations are histories, that is finite sequences of primitive actions. The successor-state axioms then express

[1] In contrast, the procedural approach encodes desired behaviors directly as program code; minimizing the role of explicit representation and reasoning can result in a much more efficient system.

the relation between two situations that are before and after the execution of an action.

The frame problem comes with two facets: A *representational* one, which concerns the efforts to specify the non-effects of actions, and an *inferential* one, which concerns the effort needed to actually compute these non-effects. Successor state axioms of the situation calculus suffice to solve the representational frame problem. The solution of the inferential frame problem can be traced to the work by Hölldobler and Schneeberger [**HS90**], based on equational logic programming, that later became known as the fluent calculus: Beside the solution to the representational frame problem that rules out the need for axioms that specify the non-effects of an action, the axiomatization in the fluent calculus that allows not applying separate inference steps for unaffected piece of knowledge provides the solution to the inferential frame problem [2].

The fluent calculus reifies the situations of the situation calculus by representing fluents as terms instead of atoms and introducing the function symbol *state*. In the fluent calculus, axioms, written in classical predicate logic, formalize equational relations between the states at consecutive situations. The original fluent calculus, which was first introduced in [**HS90**], is resource conscious, because it employs an AC1 operator in order to express the states. The interpretation of this AC1 operator is multiset union. The later version of the fluent calculus, which was introduced in [**Thi99**], treats the fluents as properties similar to the situation calculus by extending the equational theory underlying this operator with idempotency. The approach that I explore in this work is analogous to the first, resource conscious version of the fluent calculus (see Subsection 8.1.6).

The solution of the frame problem made the declarative approach to reasoning about action formalisms plausible also for planning. The GOLOG [**LRL$^+$97**] and FLUX [**Thi05**] languages, implemented in Prolog, use the expressive power of logic programming and constraint logic programming, respectively, to describe actions and plans in the lines of situation calculus and fluent calculus, respectively. [**Thi05**] provides also a comparison of these two languages.

The event calculus [**Sha99**], another popular formalism for reasoning about action, handles continuous time in a nonmonotonic circumscription (see, e.g., [**Bre91**]) based framework. In the event calculus, the term *event* is used as a synonym for the term action. Fluents hold at points in time rather than at situations and the calculus is designed to allow reasoning over intervals of time. An event calculus axiom states that a fluent is true at a point in time if the fluent was initiated by an event at some time in the past and was not terminated by an intervening event. An event may represent an action with no explicit agent.

There are also so called *action languages* for reasoning about actions in different scenarios involving different sorts of problems. Action languages have natural language like syntax and clear formal semantics. [**GL98**] is a good overview with references. [**Thi94**] and [**KT03**], respectively, establishes the relationship between the fluent calculus and the action description languages $\mathcal{A}$ and $\mathcal{A}_k$ [**GL93, LT01**], respectively.

8.1.2. Reasoning about Action and Concurrency. Most of the research in reasoning about action and change has been done under the assumption that

[2] [**Rei91**] also proposes a solution to the inferential frame problem, which is different from the solution in the fluent calculus.

an agent performs sequences of actions. In general, in the situation calculus like languages, it is assumed that the execution of an action is indivisible and uninterruptible. This is often referred to as the atomic assumption. For this reason, in such languages, when parallel execution of actions is considered, concurrency is usually defined over the parametrized time spans shared by the actions, which have durations. The focus is usually on providing a solution to those problems where an agent needs to accomplish a task that requires parallel synchronized actions, e.g., lifting a bowl of soup by holding it with both handles simultaneously.

In his PhD thesis, Pinto [**Pin94**] presents an axiomatization of the situation calculus that includes concurrent actions and continuous time by giving the starting point of an action in time as an argument of the action. [**Rei96**] further elaborates on this approach by extending the axiomatization to cover natural event occurrences that are external to the agent. These axiomatizations allow representing sets of single actions that can be executed in parallel, while lacking an operational semantics. [**DGLL00**] axiomatizes the transition semantics of the plans, which leads to an extension of the situation calculus implementation language GOLOG. However, this approach does not provide a discussion of the possible conflicts between the actions with respect to the resources being used by the actions.

The methods presented in [**Rei96**] have many similarities with the action description language $\mathcal{A}_C$ of Baral and Gelfond [**BG97**]. The action description language $\mathcal{A}_C$ extends the action description language $\mathcal{A}$ [**GL93**] to cover concurrent actions where, like in [**Rei96**], concurrent actions are represented as sets of actions that are executed simultaneously. [**GL98**] introduces a rather expressive action description language, called $\mathcal{C}$, that subsumes $\mathcal{A}_C$. The action description language $\mathcal{C}$ has a transition system semantics and can also express nondeterministic actions as well as actions with indirect effects. [**BS96**] presents a mapping of $\mathcal{A}_C$ domain descriptions into neural networks of linear size.

In [**Thi01**], Thielscher gives an account of the ideas in [**Rei96**], for concurrent continuous actions within the fluent calculus also by parameterizing time as an argument of fluents and actions.

[**KG99**] proposes an approach within temporal logic, called TAL-C, which also puts time directly into the model theory of the language and supports the description of concurrent actions with interactions. TAL-C builds on an existing logic TAL, which includes the use of dependency laws for dealing with ramification. TAL-C can represent action durations where the effects of one action interferes with or enables another action, synergistic effects of concurrent actions, conflicting and cumulative effect interactions, and resource conflicts.

8.1.3. Overview of Planners. Despite the efforts made on reasoning about action by logical methods, planners that offer effective methods and heuristics for exploring the state-space gained importance because of performance reasons. STRIPS [**FN71**] is the first major planning system that illustrates the influences of logic, state-space search and robotics. It was designed in 1971 as the planning component of the Shakey robot project at the SRI. The STRIPS language describes actions in terms of their preconditions and effects and describes the initial and goal states as conjunctions of positive literals. The effects of an action are given by so called add- and delete-lists, which are sets of fluents.

Although the action representation language of STRIPS lacks a clear logical semantics (see, e.g., [**Lif86**]), it had a strong influence on almost all the later planning systems. Bylander showed, in [**Byl92**], that simple planning in the fashion of STRIPS is PSPACE-complete. In 1986, the action description language ADL [**Ped89**] relaxed some of the restrictions in the STRIPS language, allowing disjunction, negation, and quantifiers. The Problem Domain Description Language or PDDL [**GLKM98**] was introduced as a computer-parsable, standardized syntax for representing STRIPS, ADL, and other languages. Since 1998, PDDL has been used as the standard language for the planning competitions at the AIPS conference.

Partial order planning gained importance until the mid 90s. Partial order planning (POP) algorithms explore the space of plans without committing to a totally ordered sequence of actions. They work back from the goal, adding actions to the plan to achieve each subgoal. [**PW92**] introduced the UCPOP planner, the first partial order planner for problems expressed in ADL, and provided a completeness proof. It had a better performance than its predecessors, but was rarely able to find plans with more than a dozen or so steps. Although improved heuristics were developed for UCPOP, partial order planning lost its popularity in the 1990s, and leaving its place to the Graphplan approach.

The Graphplan approach [**BM97**] is based on processing the planning graph by using a backward search to extract a plan and allowing for some partial ordering among actions. A planning graph can be constructed incrementally, starting from the initial state. Each layer contains a superset of all the literals or actions that could occur at that time step and encodes mutual exclusion relations among literals or actions that cannot co-occur. Planning graphs yield also useful heuristics for state-space-search and partial order planners. [**NK01**] give a thorough analysis of heuristics derived from planning graphs and describe a partial order planner, called REPOP, based on these ideas, which scales up much better than Graphplan.

Kautz and Selman introduced the planning-as-satisfiability approach and the SATplan algorithm, which is inspired by the success of the greedy local search for satisfiability problems. The SATplan algorithm translates a planning problem into propositional axioms and applies a satisfiability algorithm to find a model that corresponds to a valid plan. Later, several different propositional representations have been developed with varying degree of compactness and efficiency. The BLACKBOX planner combines ideas from Graphplan and SATplan [**KS98**].

The different approaches to planning have been mainly motivated by concerns about efficiency, however, so far there is no consensus on which planner and what approach is best. The most successful state-space search approach to date, at the planning competitions, is based on Hoffmann's FASTFORWARD, winner of the AIPS 2000 planning competition. This approach, combining forward and local search, uses a simplified planning graph heuristic by cutting out the branches of the search tree that do not have the potential to provide a plan. [**Hof03**] is an overview of these approaches. [**Wel94, Wel99**] are surveys of modern planning, concentrating on partial order planning, Graphplan, and SATplan.

8.1.4. Partial Order Planning and Concurrency. The atomic assumption, that considers an action indivisible, stems from the representation that only models the preconditions and effects of an action. When these preconditions and effects are seen as properties of the world, such a consideration disregards the

independence between actions: In a nutshell, two actions are independent (true-concurrent) if the effect of the two actions executed simultaneously is same as the effect of two actions being executed in isolation. Because it would be unclear from the action specification whether any two actions can be executed simultaneously without interacting with one another, an assumption that disregards the dependency between actions is not appropriate for observing parallelism in plans.

Executing actions concurrently requires explicit handling of potential resource conflicts. If two actions are left unordered, for instance, as in a partial order planning, they can be executed in either order. A partial order plan represents a set of possible totally-ordered plans. Such two unordered actions can be executed in either order, however two actions' being unordered relative to one another does not imply that they can be executed in parallel. Simultaneous execution requires that potential resource conflicts between unordered actions are made explicit and avoided by means of an approach that takes the dependency between actions into account. One of the solutions for this is to employ a formal framework that allows for explicit representation of resources, which is the approach in this thesis.

Another solution is by means of imposing resource constraints that control the order of the execution of the actions. In [**Kno94**], Knoblock presents such a method where he identifies the conditions under which two unordered actions can be executed in parallel by providing some linguistic resource constraints. Knoblock identifies the underlying assumptions about when a partial order planner can be executed in parallel, defines the class of parallel plans that can be generated by different partial order planners, and describes the changes required to turn the UCPOP planner into a parallel execution planner. In order to avoid resource conflicts, Knoblock modifies the planner to ensure that if two operators require the same resource, then they are not left unordered relative to one another.

In [**BB01**], Boutilier *et al.* exploit the ideas in [**Kno94**] and also give a good overview of the previous research on concurrent actions in reasoning about actions. There, action interactions are handled by specifying the effects of all joint (concurrent) actions directly within the formal language, which extends STRIPS in a way that can be generalized to an arbitrary planning algorithm. Boutilier *et al.* make a number of modifications to standard partial order planners by first adding equality or inequality constraints on action orderings to enforce concurrency and then expanding the definition of threads to cover concurrent actions that could prevent an intended action effect. The main addition to STRIPS representation is a concurrent action list, e.g., for an action a, that describes restrictions on other actions that can or cannot be executed concurrently with a in order for a to have the specified effect.

In [**Bäc98**], Bäckström discusses, for a class of STRIPS-like planners, reordering of plans and relaxing of a total order plan into a partial order such that actions can be executed in parallel. He proposes three different definitions with linguistic constraints for this purpose and provides a formal comparison of these different definitions. He shows that the general problem of finding an optimal parallel execution plan is NP-hard [**Bäc98**] and also compares his approach with other partial order planners.

8.1.5. Concurrency Methods in Planning. Methods of concurrency were previously considered for reasoning about actions, especially in systems for reactive planning. In reactive planning, no specific sequence of actions is planned in advance.

The planner is given a set of initial conditions and a goal. However, instead of producing a plan with branches, it produces a set of condition-action rules (see, e.g. [**Dru89**]). Along these lines, [**Ndj01**] proposes an approach in a modal logic setting for reactive systems, based on Milner's [**HM85**] observation equivalence: Within this approach, given the specification of an agent's behavior in terms of what it can do in every situation, an equivalent specification with fewer states can be derived.

Pym *et al.* propose a process algebra method for reasoning about actions in [**PPM96**] where they abstract away from the conditions and effects of actions. Ignoring the construction of plans, they describe a simple algebra of plans in order to represent actions through the processes by which changes occur. Pym *et al.* argue that the requirements of plan-execution are better met by representing actions through the processes by which changes occur than by the more widely used state-change representation. Instead of imposing a total order on plan outcomes, via some utility value, their analysis requires a partial order. Pym *et al.* define their algebra of processes, presenting and illustrating a number of combinators that allow to construct complex plans from simpler ones, and present a method to compare these plans.

Another approach based on process algebras is [**DGC96**], where the dynamic behavior of the system is modeled by a transition graph that represents all the possible system evolutions in terms of state changes caused by actions. The transition graph is defined through a description formalism in μ-calculus that leads to a process algebra model that allows to express the concurrent behavior in a multiagent dynamic system. The reasoning on the system is performed by model checking. The authors argue that, besides the features for reasoning about action, this approach inherits the tools of process algebras for dealing with complex systems, treating aspects like parallelism, communication, interruption, and coordination among agents. The approach is applicable only when complete information about the system is available.

A similar approach, where also μ-calculus is used, is presented in [**Sin98**]. However, instead of assuming a fully specified model as it is the case in [**DGC96**], this approach allows constructing a model incrementally. A branching model of time is used to express concurrent actions by multiple agents that also allows expressing the nondeterministic effects of the actions. Instead of sequences of actions, the plans are viewed as decision graphs describing the agent's actions in different situations that leads to a model of time that allows multiple future paths from each moment.

Although it is not directly related with concurrency, [**AK01**] presents an approach called *planning by rewriting* that somewhat addresses plan equivalence. The basic idea of planning by rewriting is, with respect to a plan quality measure, transforming an easy to generate, but possibly suboptimal, initial plan into a high quality plan by applying declarative plan rewriting rules in an iterative style. In planning by rewriting two plans are considered *equivalent* if they are solutions to the same planning problem, although they may differ on their cost or operators. However, the focus is not on using the rewriting rules to prove such equivalence, but using the rewriting rules to explore the solution space of plans.

8.1.6. Properties versus Resources. Traditional mathematical and logical languages, such as classical logic, are concerned with modeling properties. In a two-valued logic a property is either true or false under a given interpretation and,

in particular, cannot change its truth assignment in a reasoning episode. In this sense, being the logic of mathematical reasoning, classical logic is adequate for reasoning about unchanging entities. For instance, employing a lemma to prove a theorem does not make the lemma cease. One can use the same lemma again. This is due to the idempotent nature of conjunction and disjunction in classical logic, that is, $a \vee a \equiv a$ and $a \wedge a \equiv a$ are valid propositions in classical logic. When modeling knowledge, such a statement is very natural because there should not be any difference between knowing a once or twice.

On the other hand, the above statement makes a crucial difference when one is reasoning about action and change. As they are considered in reasoning about action, if dynamically changing worlds are to be modeled, then the truth value assigned to a particular item may change from one state to another. In fact, this phenomenon establishes the core of the frame problem, as it occurs in the situation calculus. Situation calculus proposes a solution to this problem by augmenting each fluent (atomic property) with a situation term, e.g., $open(s)$, that captures a timely behavior: The situation that results from executing an action a at that situation (successor situation) is represented by a function symbol that increments the situation term, e.g., $open(Do(a, s))$. Then, by means of the so called *frame axioms* it becomes necessary to explicitly state that every fluent that is not effected from the execution of that action is carried to the successor situation. As a result of this, the computational complexity of programs increases and the modularity of programs gets damaged.

The frame problem, which results from using atoms in classical logic to represent fluents, is an artificial problem, because it is not due to the nature of the actions and causality, but it caused by the representation scheme being used which puts properties in focus. In contrast to properties, resources, which can be consumed and produced in the course of reasoning, are a key to modeling dynamically changing worlds within a logical language, as it was first observed by Wolfgang Bibel. In [**Bib86**], Bibel introduces an approach to deductive planning where no frame axioms are needed: In the *linear connection method* he imposes a syntactical condition on proofs of the planning problems that requires each literal to engage in at most one connection.

Fluent calculus, which was first introduced in [**HS90**] as the equational logic programming paradigm, brings the resources under the realm of classical logic. It is based on the idea of reifying fluents and states, and representing them on the term level by using an associative commutative function symbol that admits a unit element and is not idempotent [3]. The non-idempotency is the key to view fluents as resources that can be produced and consumed in a reasoning episode. It is also the key for solving the technical frame problem representationally as well as computationally (inferential frame problem). Specifying conjunctions of fluents using an AC1-operator essentially amounts to defining the states over the data structure multiset and considering actions as multiset rewriting rules. Fluents represent

[3][**Thi99**] presents a different version of the fluent calculus, where the AC1 operator is extended with idempotency and this way fluents are treated as properties. However, the extensions of this approach that propose solutions to *sensing, concurrency, ramification,* and *qualification problems* lack modularity and a unifying operational semantics, in the sense that these solutions can be integrated with each other without going through an intrinsic engineering process.

resources that are consumed and produced by the actions. The fluent calculus, introduced in [**HS90**], give a more general account of such planning problems which are called *conjunctive planning problems*.

The equational axioms for the AC1 equational theory, in the fluent calculus, are built into the unification computation and SLDE-resolution is applied as an inference rule. SLDE-resolution extends Prolog's SLD-resolution with an equational theory of a function symbol. Expressing conjunctive planning problems within classical logic requires an additional machinery to the logic that takes care of unification under the AC1 theory. The linear logic approach to deductive planning, introduced in [**MTV90**], does not require such an additional unification mechanism and suffices to give the full operational semantics of the planning problems.

Linear logic [**Gir87**] was created in 1987 as a logic motivated by languages for concurrent and resource-oriented computations and revived interest in all the related substructural logics, which in the past were mainly studied by philosophers without having computation in mind. In the mean time, linear logic has been studied by many researchers and has been broadly recognized as a logic of concurrency (see, e.g., [**Mil92, EW93**]). In linear logic, weakening and contraction rules of classical logic that cause the idempotent behavior of conjunction and disjunction are controlled: The multiplicative conjunction $\otimes$ is not idempotent, that is, "$a \otimes a \vdash a$" is not provable. This allows linear logic to easily represent actions and causality as it is considered in the linear connection method and fluent calculus. In fact, resource conscious fluent calculus, Bibel's linear connection method, and the linear logic approach to planning are equivalent, as it was shown in [**GHS96**].

8.2. Linear Logic Planning

The inference rule *modus ponens* of classical logic infers q from p and $p \Rightarrow q$. Similar to the way people reason, after the conclusion q is reached, the premises p and $p \Rightarrow q$ are remembered (preserved) to be used in a later inference. Such a process of acquiring knowledge, as it is modeled in classical logic, is cumulative. This cumulative behavior is an essential feature of mathematical language: When people are gaining new knowledge, with the help of books and taking notes, they *ideally* do not forget the previous knowledge on which the new knowledge is built.

Linear logic interprets this syntactical inference quite differently. Viewing p and q as resources being consumed and produced, once the linear implication $p \multimap q$ and p is used in an inference, they are used up, hence become unavailable for another inference. For instance, in the context of messages, which are sent and received by processes, the implication $p \multimap q$ models that upon receiving the message p, message q is sent out, but there is no remembering of p, which is a typical situation, e.g., in a communication protocol where the information is not archived.

In the linear logic approach to planning, the formula $p \multimap q$ is interpreted as an action in the context of a conjunctive planning problem. In this case its application transforms a state p into a state q. These states are treated as resources, that is, when action $p \multimap q$ is applied to the state p, p is consumed (annihilated) and state q is produced. This results in a representation of change, called conjunctive planning, elegantly solving the frame problem.

8.2.1. Conjunctive Planning Problems. Before introducing linear logic planning, let me recall the basic definitions of conjunctive planning, following [**GHS96, MTV90**]:

NOTATION 8.1. *Multisets are denoted by the curly brackets "$\dot\{$" and "$\dot\}$". The empty multiset is denoted by $\dot\emptyset$. $\dot\cup$, $\dot-$, and $\dot\subseteq$ denote the multiset operations corresponding to the usual set operations $\cup$, $-$, and $\subseteq$, respectively.*

DEFINITION 8.2. *A* conjunctive planning domain *is given by:*

i. a finite set $\mathscr{R}$ of constants, which represent atomic properties of the world and are called resources*. Resources are denoted by $a, b, c, \ldots$;*

ii. a finite set $\mathscr{A}$ of actions (transition rules) *of the form*

$$\mathsf{a} \;:\; \dot\{\, c_1, \ldots, c_p \,\dot\} \to \dot\{\, e_1, \ldots, e_q \,\dot\},$$

where a *is the* name *of the action and $\dot\{\, c_1, \ldots, c_p \,\dot\}$ and $\dot\{\, e_1, \ldots, e_q \,\dot\}$ are multisets of resources, which are called* condition *and* effect*, respectively.*

DEFINITION 8.3. *Given a set $\mathscr{R}$ of resources, a* world state*, denoted by $\mathcal{Z}$, $\mathcal{I}$, or $\mathcal{G}$, is a multiset of resources from $\mathscr{R}$. I will use the word 'state' instead of 'world state' where no confusion is possible.*

DEFINITION 8.4. *Given a planning domain with $\mathscr{R}$ and $\mathscr{A}$, a conjunctive planning problem $\mathscr{P}$ is given by $\langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{I}$ and $\mathcal{G}$ are two distinguished states, which are called the* initial state *and the* goal state*, respectively.*

Now, to illustrate the above definitions, let us see the following example, which is a modification of an example from [**GHS96**].

EXAMPLE 8.5. *Suppose Peter is working on a Sunday at the computer science department and he feels hungry. Peter is an easy-going person, so he will be happy (h) if he gets a candy-bar (c) and also a lemonade (l) to go with it. There is a vending machine in the department, which offers both the lemonade (l) and the candy-bar (c). The lemonade and the candy-bar cost 50 cents (f) each. Peter has a euro (e) in his pocket. However, because the vending machine accepts only 50 cents coins, he has to get change for his euro. This scenario can be described as the planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$ with*

$$\mathscr{R} = \{h, c, l, e, f\}$$

and the set $\mathscr{A}$ of actions that contains

$$\begin{array}{llll} \mathsf{c}_{\mathsf{euro}} : & \dot\{\, e \,\dot\} & \to & \dot\{\, f, f \,\dot\}, \\ \mathsf{b}_{\mathsf{candy}} : & \dot\{\, f \,\dot\} & \to & \dot\{\, c \,\dot\}, \end{array} \qquad \begin{array}{llll} \mathsf{b}_{\mathsf{lem}} : & \dot\{\, f \,\dot\} & \to & \dot\{\, l \,\dot\}, \\ \mathsf{h}_{\mathsf{lunch}} : & \dot\{\, c, l \,\dot\} & \to & \dot\{\, h \,\dot\} \end{array}$$

respectively, that allow him to change a euro for two 50 cents coins, buy a candy-bar, buy a lemonade, and have lunch, respectively.

DEFINITION 8.6. *An action* a *of the form*

$$\dot\{\, c_1, \ldots, c_p \,\dot\} \;\to\; \dot\{\, e_1, \ldots, e_q \,\dot\},$$

is applicable in a state $\mathcal{Z}$ if and only if $\dot\{\, c_1, \ldots, c_p \,\dot\} \mathrel{\dot\subseteq} \mathcal{Z}$. The application of an action a *to a state $\mathcal{Z}$ is defined by the function Φ, where it is applicable, as*

$$\Phi(\mathsf{a}, \mathcal{Z}) = (\, \mathcal{Z} \mathbin{\dot-} \dot\{\, c_1, \ldots, c_p \,\dot\} \,) \mathbin{\dot\cup} \dot\{\, e_1, \ldots, e_q \,\dot\}.$$

DEFINITION 8.7. *A* plan *is a structure generated by*

$$\mathsf{P} ::= \circ \mid \mathsf{a} \mid \langle\, \mathsf{P} \,;\, \mathsf{P} \,\rangle$$

where a *denotes atoms representing actions and* $\circ$ *denotes the empty plan. As before, the operator* $\langle \underline{\cdot}\,;\underline{\cdot}\rangle$ *is associative and* $\circ$ *is its left and right unit. Plans are assumed to be in unit normal form, that is, there are no occurrences of* $\circ$ *in a plan that can be equivalently removed. The* length of a plan *is the number of actions in that plan.*

DEFINITION 8.8. *The* application of a plan $\mathsf{P} = \langle \mathsf{a}_1; \ldots; \mathsf{a}_k\rangle$ to a state $\mathcal{Z}_0$ *is defined as*

$$\Phi(\mathsf{a}_k, \ldots, \Phi(\mathsf{a}_1, \mathcal{Z}_0)\ldots) = \mathcal{Z}$$

where $\mathcal{Z}$ *is the resulting state. If it is more convenient,* $\Phi(\mathsf{a}_k, \ldots, \Phi(\mathsf{a}_1, \mathcal{Z}_0)\ldots)$ *will be abbreviated with* $\Phi(\mathsf{P}, \mathcal{Z}_0)$*. For a planning problem* $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G}\rangle$*, a plan* P solves $\mathscr{P}$ *if* $\Phi(\mathsf{P}, \mathcal{I}) = \mathcal{G}$*.*

EXAMPLE 8.9. *Consider the planning problem of Example 8.5. Clearly, two solutions of this planning problem are the following two plans:*

$$\langle \mathsf{c}_{\mathsf{euro}}\,;\, \mathsf{b}_{\mathsf{candy}}\,;\, \mathsf{b}_{\mathsf{lem}}\,;\, \mathsf{h}_{\mathsf{lunch}}\rangle \qquad\qquad \langle \mathsf{c}_{\mathsf{euro}}\,;\, \mathsf{b}_{\mathsf{lem}}\,;\, \mathsf{b}_{\mathsf{candy}}\,;\, \mathsf{h}_{\mathsf{lunch}}\rangle$$

These two plans differ in the order of execution of the actions $\mathsf{b}_{\mathsf{lem}}$ *and* $\mathsf{b}_{\mathsf{candy}}$*. Thus, when all the plans solving this problem are considered, these two actions are partially ordered in the sense that they can be executed in either order. In fact, due to the explicit representation of resources by means of multisets, without committing to a totally ordered plan, such partially ordered actions can also be executed simultaneously without causing any resource conflicts.*

PROPOSITION 8.10. *For every planning problem* $\mathscr{P}$ *given with the initial state* $\mathcal{I}$*, the goal state* $\mathcal{G}$*, the set* $\mathscr{A}$ *of actions, and a plan* $\langle a_1; \ldots; a_k\rangle$ *that solves* $\mathscr{P}$*, for any* $s \leq k$*, there is a planning problem* $\mathscr{P}'$ *given with the initial state* $\mathcal{I}' = \Phi(a_s, \ldots, \Phi(a_1, \mathcal{I})\ldots)$*, goal* $\mathcal{G}$*, and* $\mathscr{A}$ *that is solved by* $\langle a_{s+1}; \ldots; a_k\rangle$*.*

PROOF. Given that $\langle a_1; \ldots; a_s; a_{s+1}; \ldots; a_k\rangle$ solves $\mathscr{P}$, it follows that $\mathcal{I}' = \Phi(a_s, \ldots, \Phi(a_1, \mathcal{I})\ldots)$ is a state and $\Phi(a_k, \ldots, \Phi(a_{s+1}, \mathcal{I}')\ldots) = \mathcal{G}$. □

PROPOSITION 8.11. *For any states* $\mathcal{I}$*,* $\mathcal{Z}_1$*,* $\mathcal{Z}_2$*, and plan* P*, if* $\Phi(\mathsf{P}, \mathcal{I}) = \mathcal{Z}_1$ *then* $\Phi(\mathsf{P}\,, \mathcal{I} \dot\cup \mathcal{Z}_2) = \mathcal{Z}_1 \dot\cup \mathcal{Z}_2$*.*

PROOF. With induction on the length of the plan P. If P is the empty plan then we have $\Phi(\circ\,, \mathcal{I} \dot\cup \mathcal{Z}_2) = \mathcal{I} \dot\cup \mathcal{Z}_2$. Turning to the inductive step, assume that the proposition holds for a plan of length k. Let $\langle \mathsf{P}; \mathsf{a}\rangle$ be a plan of length $k+1$, where $\mathcal{C}$ and $\mathcal{E}$ are the condition and effect of the action a, respectively. Assume that $\Phi(\mathsf{a}\,, \Phi(\mathsf{P}, \mathcal{I})) = \mathcal{Z}_1$ and $\Phi(\mathsf{P}, \mathcal{I}) = \mathcal{Z}$. It follows that $\mathcal{Z}_1 = (\mathcal{Z} \dot- \mathcal{C}) \dot\cup \mathcal{E}$. From the induction hypothesis, we have $\Phi(\mathsf{a}\,, \Phi(\mathsf{P}, \mathcal{I} \dot\cup \mathcal{Z}_2)) = \Phi(\mathsf{a}\,, \mathcal{Z} \dot\cup \mathcal{Z}_2)$. Because a is applicable in $\mathcal{Z}'$ it is also applicable in $\mathcal{Z}' \dot\cup \mathcal{Z}_2$. Thus, $\Phi(\mathsf{a}, \mathcal{Z}' \dot\cup \mathcal{Z}_2) = ((\mathcal{Z}' \dot\cup \mathcal{Z}_2) \dot- \mathcal{C}) \dot\cup \mathcal{E} = \mathcal{Z}_1 \dot\cup \mathcal{Z}_2$. □

8.2.2. Linear Logic Approach to Conjunctive Planning. I will now briefly revise the mapping of the conjunctive planning problems to multiplicative fragment of linear logic, as can be found in [**MTV90**]:

Formulas in the multiplicative fragment of linear logic that are used in [**MTV90**] are either atoms or of the form $s \otimes t$, where s and t are formulas. If not stated otherwise, s, t, ... denote formulas and $\Gamma, \Delta, \ldots$ denote multisets of formulas:

$$axiom\ \frac{}{a \vdash a} \qquad (cut)\ \frac{\Gamma \vdash s \quad \Delta,\, s \vdash t}{\Gamma, \Delta \vdash t} \qquad 1_r\ \frac{}{\vdash 1}$$

$$(\otimes_l)\ \frac{\Gamma, s, t \vdash u}{\Gamma, s \otimes t \vdash u} \qquad (\otimes_r)\ \frac{\Gamma \vdash s \quad \Delta \vdash t}{\Gamma, \Delta \vdash s \otimes t} \qquad (1_l)\ \frac{\Gamma \vdash s}{\Gamma, 1 \vdash s}$$

FIGURE 8.1. Conjunctive linear theory

DEFINITION 8.12. *A* conjunctive linear theory *consists of the axioms and rules in Figure 8.1 together with the* proper axioms

$$\mathsf{a}\ \frac{}{c_1, \ldots, c_p \vdash e_1 \otimes \ldots \otimes e_q}$$

for each action of the form $\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \to \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$. *Sequents are limited to only one formula on the right-hand-side, in the spirit of intuitionistic logic.*

A proof of a conjunctive planning problem $\Gamma \vdash t$ is a solution for a conjunctive planning problem if and only if Γ is the multiset that represents the initial state, and t is the multiplicative conjunction of the atoms representing resources that are available in the goal state.

EXAMPLE 8.13. *To illustrate these ideas, let us consider the planning problem in Example 8.5. We represent these actions as the following proper axioms*

$$\mathsf{c_{euro}}\ \frac{}{e \vdash f \otimes f} \qquad \mathsf{b_{lem}}\ \frac{}{f \vdash l} \qquad \mathsf{b_{candy}}\ \frac{}{f \vdash c} \qquad \mathsf{h_{lunch}}\ \frac{}{l, c \vdash h}$$

and the planning problem as $e \vdash h$. *Then we get the following proof:*

$$\dfrac{\mathsf{c_{euro}}\,\dfrac{}{e \vdash f \otimes f} \qquad \dfrac{\dfrac{\mathsf{b_{lem}}\,\dfrac{}{f \vdash l} \qquad \dfrac{\mathsf{b_{candy}}\,\dfrac{}{f \vdash c} \quad \mathsf{h_{lunch}}\,\dfrac{}{c\,,\, l \vdash h}}{l\,,\, f \vdash h}\,cut}{f\,,\, f \vdash h}}{f \otimes f \vdash h}\,\otimes_l}{e \vdash h}\,cut$$

In this approach, the plan is extracted by reading the leaves of the proof tree from left to right. For instance, the above proof reads as the plan

$$\langle \mathsf{c_{euro}}\ ;\ \mathsf{b_{lem}}\ ;\ \mathsf{b_{candy}}\ ;\ \mathsf{h_{lunch}} \rangle\ ,$$

which is one of the solutions of this planning problem. For this reason, while constructing the proof, we must keep track of the premises: In an application of a cut rule, the sequents $\Gamma \vdash s$ and $\Delta, s \vdash t$ must occur, respectively, on the left and right side of the premise. Furthermore, in an application of the $\otimes_r$ rule one has to decide which premise is written to the left and which one to the right nondeterministically. These decisions disregard the other plans that solve the same

planning problem, and in particular any possible partial order between the actions remains hidden.

In the following, I will present a different encoding of the conjunctive planning problems in the multiplicative exponential linear logic. This encoding will allow to construct cut-free proofs and extract partial order plans from proofs that have a non-interleaving concurrency semantics.

This new encoding can be presented in the sequent calculus presentation of multiplicative exponential linear logic. However, I will employ the calculus of structures presentation of multiplicative exponential linear logic, namely system ELS. There are two reasons for this:

(1) In section 8.4, I present an encoding of conjunctive planning problems in system NEL. System NEL cannot be expressed in the sequent calculus. However, the structures and inference rules of systems NEL and ELS are quite similar. Using the same formalism, i.e., the calculus of structures, for both encodings is helpful to carry the results in both directions.
(2) I will use some proof theoretical properties of system ELS such as decomposition of ELS proofs. These properties are not available in the sequent calculus presentation of multiplicative exponential linear logic.

8.2.3. Conjunctive Planning Problems in System ELS. I will now present an encoding of the planning problems in the language of ELS, that is, in multiplicative exponential linear logic in the calculus of structures. Instead of using proper axioms for actions, I embed the actions in a structure that represents the planning problem. This way, we will observe cut-free proofs. Beside the nondeterminism due to the choice of the competing actions, the availability of the cut rule brings an extra nondeterminism: In a bottom-up search, applying a cut rule means guessing a formula to be appropriate to be the cut formula that will result in a proof. By having a cut-free proof system, we reduce this nondeterminism in proof search to the choice of the application of the inference rules.

Consider the following sequent calculus encoding of an action $\dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$.

$$c_1 \otimes \ldots \otimes c_p \multimap e_1 \otimes \ldots \otimes e_q$$

Note that *linear implication* $\multimap$ is defined as $s \multimap t = s^{\perp} \parr t$, where $\parr$ is a par connective. $\otimes$ is the *times* connective. This way we obtain an action as a linear logic formula, analogous to the proper axioms representing actions. However, because the encoding is in a one-sided calculus, I take the De Morgan dual of this formula. Let us see this as an ELS structure:

Definition 8.14. *Given an action* $\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$*, the* conjunctive action structure for a*, denoted by* A *(possibly indexed), is a structure of the form*

$$(\bar{c}_1, \ldots, \bar{c}_p, [e_1, \ldots, e_q]) \,.$$

The initial and goal states of a planning problem are encoded similarly:

Definition 8.15. *Given an initial state* $\mathcal{I} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$ *and a goal state* $\mathcal{G} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$*, the* problem structure for $\mathcal{I}$ *and* $\mathcal{G}$*, denoted by* K*, is a structure of the form*

$$[r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)] \,.$$

This way, in an abstract logic programming setting, this encoding allows to observe an explicit logical duality between the problem and action structures:

$$\overline{(\bar{c}_1, \ldots, \bar{c}_p, [e_1, \ldots, e_q])} \quad = \quad [c_1, \ldots, c_p, (\bar{e}_1, \ldots, \bar{e}_q)]$$

We are now ready to define a conjunctive planning problem in the language of ELS.

DEFINITION 8.16. *Given a conjunctive planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$, let $\mathsf{A}_1, \ldots, \mathsf{A}_\mathsf{s}$ be the action structures for all the actions $\mathsf{a}_1, \ldots, \mathsf{a}_s \in \mathscr{A}$ and K be the problem structure for $\mathcal{I}$ and $\mathcal{G}$.* The conjunctive planning problem structure (cpps) for $\mathscr{P}$*, denoted by $\mathcal{P}$, is defined as follows:*

$$[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,\mathsf{K}]$$

In the above encoding, because an action can be executed arbitrarily many times, I employ '?' of linear logic, which retains a controlled contraction and weakening on the action structures. This way, an action structure can be duplicated, when needed, by applying the $\mathsf{b}{\downarrow}$ rule, or annihilated by applying the rule $\mathsf{w}{\downarrow}$ during the proof search. To make the interaction between the planning problems and actions explicit, I prefix a planning problem with '!'. This allows an action structure to get inside and interact with a problem structure by an application of the rule $\mathsf{p}{\downarrow}$. Because of the duality between '?' and '!', the duality between action structures and the problem structures remains preserved.

EXAMPLE 8.17. *The cpps for the planning problem of Example 8.5 is as follows:*

$$[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[e, \bar{h}]\,]$$

The structures $(\bar{e}, [f, f])$, $(\bar{f}, l)$, $(\bar{f}, c)$, and $(\bar{l}, \bar{c}, h)$, respectively, are the conjunctive action structures for the actions c_euro, b_lem, b_can, and $\mathsf{h}_\mathsf{lunch}$, respectively. The atom e denotes the initial state and the atom $\bar{h}$ denotes the goal state.

I will now show that proving a cpps in ELS is equivalent to showing that the corresponding conjunctive planning problem has a solution. I will first need some definitions and lemmas.

DEFINITION 8.18. *The following rule is called* action.

$$\mathsf{action}\ \frac{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[E, R]\,]}{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[c_1, \ldots, c_p, R]\,]}$$

LEMMA 8.19. *The rule* action *is derivable for system* ELS.

PROOF. Take the following derivation where the instance of the rule $\mathsf{i}{\downarrow}$ is as given in Proposition 4.46:

$$\dfrac{\dfrac{\dfrac{\dfrac{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[E, R]\,]}{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[([(\bar{c}_1, \ldots, \bar{c}_p)\,, c_1, \ldots, c_p], E), R]\,]}\,\mathsf{i}{\downarrow}}{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[(\bar{c}_1, \ldots, \bar{c}_p, E)\,, c_1, \ldots, c_p, R]\,]}\,\mathsf{s}}{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[c_1, \ldots, c_p, R]\,]}\,\mathsf{p}{\downarrow}}{S[\,?(\bar{c}_1, \ldots, \bar{c}_p, E)\,,\,!\,[c_1, \ldots, c_p, R]\,]}\,\mathsf{b}{\downarrow}$$

□

DEFINITION 8.20. *The following rule is called* termination.

$$\mathsf{termination}\ \frac{}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, !\,[g_1, \ldots, g_m, (\bar{g}_1, \ldots, \bar{g}_m)]]}$$

LEMMA 8.21. *The rule* termination *is derivable for system* ELS.

PROOF. Take the following derivation where the instance of the rule i↓ is as given in Proposition 4.46:

$$\mathsf{w}\downarrow\ \frac{\mathsf{w}\downarrow\ \dfrac{\mathsf{i}\downarrow\ \dfrac{1\downarrow\ \dfrac{}{1}}{!\,[g_1, \ldots, g_m, (\bar{g}_1, \ldots, \bar{g}_m)]}}{\vdots}}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, !\,[g_1, \ldots, g_m, (\bar{g}_1, \ldots, \bar{g}_m)]]}$$

□

It is important to observe that the inference rules action and termination provide the operational semantics of a planner, that is, these inference rules can be used as machine instructions in an implementation of this approach. Let us see these rules on our example:

EXAMPLE 8.22. *A proof of the cpps for the planning problem of Example 8.5 can be constructed as follows:*

$$\mathsf{termination}\ \frac{\mathsf{action}\ \dfrac{\mathsf{action}\ \dfrac{\mathsf{action}\ \dfrac{\mathsf{action}\ \dfrac{\mathsf{termination}\ \dfrac{}{[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[h, \bar{h}]\,]}}{[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[l, c, \bar{h}]\,]}}{[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[l, f, \bar{h}]\,]}}{[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[f, f, \bar{h}]\,]}}{[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), !\,[e, \bar{h}]\,]}$$

Now, let me state the following theorem, which Straßburger proved in [**Str03b**]. It is important to note that it is not possible to state an analogous result in the sequent calculus presentation of multiplicative exponential linear logic:

THEOREM 8.23. (decomposition) *For every proof* $\prod_R$ *in system* ELS*, there are derivations* $\Delta_1, \ldots, \Delta_4$*, such that*

$$\begin{array}{c} \Delta_4 \Big\| \{\mathsf{ai}\downarrow\} \\ R_3 \\ \Delta_3 \Big\| \{\mathsf{s}, \mathsf{p}\downarrow\} \\ R_2 \\ \Delta_2 \Big\| \{\mathsf{w}\downarrow\} \\ R_1 \\ \Delta_1 \Big\| \{\mathsf{b}\downarrow\} \\ R \end{array}$$

for some structures R_1,R_2*, and* R_3*.*

With the help of the following lemma, this decomposition theorem provides a decomposition of the proofs of the cpps where only a single inference rule is used at each phase of the proof. In the following, this decomposition of the proofs of the cpps will be useful in proving some properties of the cpps.

LEMMA 8.24. *The rule* s *permutes over the rule* $\mathsf{p}\downarrow$ *if the redex of* $\mathsf{p}\downarrow$ *is not inside an active structure of the contractum of* s.

PROOF. It suffices to check the only case excluded by the conditions of Remark 5.49, that is, when the contractum of s is inside an active structure of the redex of $\mathsf{p}\downarrow$. In this case, we permute as follows:

$$\mathsf{p}\downarrow\dfrac{\mathsf{s}\dfrac{S\{![P,([R,U],T)]\}}{S[!P,?([R,U],T)]}}{S[!P,?[(R,T),U]]} \quad\rightsquigarrow\quad \mathsf{s}\dfrac{\mathsf{p}\downarrow\dfrac{S\{![P,([R,U],T)]\}}{S[!P,?[(R,T),U]]}}{S\{![P,(R,T),U]\}}$$

$$\mathsf{p}\downarrow\dfrac{\mathsf{s}\dfrac{S\{![([R,U],T),P]\}}{S[!([R,U],T),?P]}}{S[![(R,T),U],?P]} \quad\rightsquigarrow\quad \mathsf{s}\dfrac{\mathsf{p}\downarrow\dfrac{S\{![([R,U],T),P]\}}{S[![(R,T),U],?P]}}{S\{![(R,T),U,P]\}}$$

□

By using this result, we can achieve a finer decomposition of the proofs of the conjunctive planning problem structures compared to Theorem 8.23:

COROLLARY 8.25. *Let* $\mathcal{P} = [?\mathsf{A}_1,\ldots,?\mathsf{A}_\mathsf{s},!\,\mathsf{K}]$ *be a cpps. For every proof* Π *of* $\mathcal{P}$ *in system* ELS*, there are derivations* $\Delta_1,\ldots,\Delta_5$*, such that*

$$\begin{array}{c}
\Delta_5 \Big\Vert \{\mathsf{ai}\downarrow\} \\
!([a_1,\bar{a}_1],\ldots,[a_m,\bar{a}_m]) \\
\Delta_4 \,\|\, \{\mathsf{s}\} \\
![\mathsf{A}_1,\ldots,\mathsf{A}_\mathsf{k},\mathsf{K}] \\
\Delta_3 \,\|\, \{\mathsf{p}\downarrow\} \\
[?\mathsf{A}_1,\ldots,?\mathsf{A}_\mathsf{k},!\,\mathsf{K}] \\
\Delta_2 \,\|\, \{\mathsf{w}\downarrow\} \\
[?\mathsf{A}_1,\ldots,?\mathsf{A}_\mathsf{s},?\mathsf{A}_{\mathsf{s}+1},\ldots,?\mathsf{A}_\mathsf{n},!\,\mathsf{K}] \\
\Delta_1 \,\|\, \{\mathsf{b}\downarrow\} \\
[?\mathsf{A}_1,\ldots,?\mathsf{A}_\mathsf{s},!\,\mathsf{K}]
\end{array}$$

where for all $\mathsf{A} \in \dot{\{}\,\mathsf{A}_1,\ldots,\mathsf{A}_\mathsf{s},\mathsf{A}_{\mathsf{s}+1},\ldots,\mathsf{A}_\mathsf{n}\,\dot{\}}$*, it holds that* $\mathsf{A} \in \{\mathsf{A}_1,\ldots,\mathsf{A}_\mathsf{s}\}$ *and there are* k *number of instances of the rule* $\mathsf{p}\downarrow$.

PROPOSITION 8.26. *Let* $R = [S\{\bar{a}\},a]$ *be an* ELS *structure that consists of pairwise distinct atoms.* R *has a proof in system* $\{\mathsf{ai}\downarrow,\mathsf{s}\}$ *if and only if* $S\{1\}$ *has a proof.*

PROOF. ($\Rightarrow$:) Construct a proof of $S\{1\}$ from the proof of R by replacing a with $\bot$ and $\bar{a}$ with 1. ($\Leftarrow$:) The proof follows from the derivation

$$\begin{array}{c}
\mathsf{ai}\downarrow\dfrac{S\{1\}}{S[\bar{a},a]} \\
\Big\Vert \{\mathsf{s}\} \\
[S\{\bar{a}\},a]
\end{array}\ .$$

□

THEOREM 8.27. *Let $\mathscr{P}$ be a planning problem and $\mathcal{P}$ be the cpps for $\mathscr{P}$. There is a plan P with length k that solves $\mathscr{P}$ if and only if there is a proof of $\mathcal{P}$ with k number of instances of the rule $\mathsf{p}\!\downarrow$.*

PROOF. Proof by induction on k.
($\Rightarrow$:) For the base case, if P is the empty plan then it must be that

$$\mathcal{I} = \dot{\{}\, g_1, \ldots, g_m \,\dot{\}} = \mathcal{G}\,.$$

Together with Lemma 8.21, take the proof

$$\mathsf{termination}\,\frac{}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,[g_1, \ldots, g_m, (\bar{g}_1, \ldots, \bar{g}_m)]]}\quad .$$

For the induction step we assume that the result holds for a plan with k actions. Suppose there is a planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G}\rangle$ where

$$\mathcal{I} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}\,, \qquad \mathcal{G} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}\ .$$

Assume that $\langle \mathsf{a}_1; \ldots; \mathsf{a}_\mathsf{k}; \mathsf{a}_{\mathsf{k}+1}\rangle$ solves the planning problem $\mathscr{P}$. Then we find an action $\mathsf{a}_1 \in \mathscr{A}$ and a planning problem $\mathscr{P}' = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}', \mathcal{G}\rangle$ such that

$$\mathsf{a}_1 : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \to \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}\,,$$

$$\mathcal{I}' = \dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}} = (\,\dot{\{}\, r_1, \ldots, r_m \,\dot{\}} \mathbin{\dot{-}} \dot{\{}\, c_1, \ldots, c_p \,\dot{\}}\,) \mathbin{\dot{\cup}} \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$$

and the plan $\langle \mathsf{a}_2; \ldots; \mathsf{a}_\mathsf{k}; \mathsf{a}_{\mathsf{k}+1}\rangle$ solves $\mathscr{P}'$. With the induction hypothesis we find

$$\mathcal{P} = [?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, ![r_1{}', \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_q)]] \quad \text{such that} \quad \underset{\mathcal{P}}{\Pi \prod \mathsf{ELS}}\ .$$

Together with the Lemma 8.19 take the following proof:

$$\mathsf{action}\,\frac{\begin{array}{c}\Pi \prod \mathsf{ELS} \\ [?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,[r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)]]\end{array}}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,[r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]]}$$

($\Leftarrow$:) For the base case, if there are non applications of the rule $\mathsf{p}\!\downarrow$ in Π, then from Corollary 8.25, there must be a decomposition of Π as follows:

$$\begin{array}{c} \Pi' \prod_{\{\mathsf{s},\mathsf{ai}\downarrow\}} \\ ![r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)] \\ \|_{\{\mathsf{w}\downarrow\}} \\ [?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, ![r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]] \end{array}$$

In order for a proof Π' to exist, it must be that $\dot{\{}\, r_1, \ldots, r_m \,\dot{\}} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$. Thus, there is a plan with length 0 that solves the planning problem $\mathscr{P}$.

For the induction step we assume that the result holds for a proof with k number of instances of the rule $\mathsf{p}\!\downarrow$. Suppose that there is a planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G}\rangle$ where

$$\mathcal{I} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}\,, \qquad \mathcal{G} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}\ ,$$

$\mathcal{P}$ is its encoding and there is a proof of $\mathcal{P}$ with $k+1$ number of instances of the rule $\mathsf{p}\downarrow$. From Corollary 8.25, there must be a decomposition of Π, with $k+1$ instances of the rule $\mathsf{p}\downarrow$, as follows:

$$\begin{array}{c}
\Pi' \left\| \{\mathsf{ai}\downarrow,\mathsf{s}\} \right. \\
!\,[\mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{A}_{k+1}, \mathsf{K}] \\
\Delta_3 \,[\!]\, \{\mathsf{p}\downarrow\} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_k, ?\mathsf{A}_{k+1}, !\,\mathsf{K}] \\
\Delta_2 \,[\!]\, \{\mathsf{w}\downarrow\} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, ?\mathsf{A}_{s+1}, \ldots, ?\mathsf{A}_n, !\,\mathsf{K}] \\
\Delta_1 \,[\!]\, \{\mathsf{b}\downarrow\} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, !\,\mathsf{K}]
\end{array}$$

Let Π'' be the following proof obtained from Π' by renaming the atoms in Π' in a way such that there are only structures that consist of pairwise distinct atoms at the premise and conclusion of each instance of the inference rules.

$$\begin{array}{c}
\Pi'' \left\| \{\mathsf{ai}\downarrow,\mathsf{s}\} \right. \\
[\mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{A}_{k+1}, r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]
\end{array}$$

Thus, for every $r \in \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$, there must be an action structure

$$\mathsf{A} = (\bar{c}_1, \ldots, \bar{c}_p, [e_1, \ldots, e_q]) \in \dot{\{}\, \mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{A}_{k+1} \,\dot{\}}$$

such that $r \in \dot{\{}\, c_1, \ldots, c_p \,\dot{\}}$. (Without loss of generality assume that $r \notin \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$). Since there cannot be an ELS structure of the form

$$[(\bar{c}_{1,1}, \ldots, \bar{c}_{1,p_1}, [e_{1,1}, \ldots, e_{1,q_1}]), \ldots, (\bar{c}_{n,1}, \ldots, \bar{c}_{n,p_n}, [e_{1,1}, \ldots, e_{1,q_n}]), (\bar{g}_1, \ldots, \bar{g}_s)]$$

which is provable in system $\{\mathsf{ai}\downarrow, \mathsf{s}\}$, it follows from Proposition 8.26 that there must be an action structure $\mathsf{A} \in \dot{\{}\, \mathsf{A}_1, \ldots, \mathsf{A}_k, \mathsf{A}_{k+1} \,\dot{\}}$ such that

$$\mathsf{A} = (\bar{c}_1, \ldots, \bar{c}_p, [e_1, \ldots, e_q]) \quad \text{and} \quad \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \;\dot{\subseteq}\; \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}\,.$$

Then there must be an action $\mathsf{a} \in \mathscr{A}$ such that

(1) $$\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \to \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}\,.$$

(2) Let $\dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}} = (\,\dot{\{}\, r_1, \ldots, r_m \,\dot{\}} \mathbin{\dot{-}} \dot{\{}\, c_1, \ldots, c_p \,\dot{\}}\,) \,\dot{\cup}\, \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}\,.$

Because of commutativity and associativity we can assume that $\mathsf{A} = \mathsf{A}_{k+1}$. By applying Proposition 8.26 we get the following proof:

$$\begin{array}{c}
\Pi_1 \left\| \{\mathsf{ai}\downarrow,\mathsf{s}\} \right. \\
[\mathsf{A}_1, \ldots, \mathsf{A}_k, r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)] \\
[\!]\, \{\mathsf{ai}\downarrow,\mathsf{s}\} \\
[\mathsf{A}_1, \ldots, \mathsf{A}_k, (c_1, \ldots, c_p, [e_1, \ldots, e_q]), r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]
\end{array}$$

With Corollary 8.25 and proof Π_1 above, we can construct a proof, with k number of applications of the rule $\mathsf{p}\downarrow$, of the cpps $\mathcal{P}'$ for the planning problem $\mathscr{P}' =$

$\langle \mathscr{R}, \mathscr{A}, \mathcal{I}', \mathcal{G} \rangle$, where $\mathcal{I}' = \dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}}$ as follows:

$$
\begin{array}{c}
\Pi_1 \prod\!\!\!\prod_{\{\mathsf{ai}\downarrow,\mathsf{s}\}} \\
![\mathsf{A}_1, \ldots, \mathsf{A}_k, r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)] \\
[\!|_{\{\mathsf{p}\downarrow\}} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_k, !\,[r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)]] \\
[\!|_{\{\mathsf{w}\downarrow\}} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, ?\mathsf{A}_{s+1}, \ldots, ?\mathsf{A}_n, !\,[r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)]] \\
[\!|_{\{\mathsf{b}\downarrow\}} \\
[?\mathsf{A}_1, \ldots, ?\mathsf{A}_s, !\,[r'_1, \ldots, r'_{m'}, (\bar{g}_1, \ldots, \bar{g}_n)]]
\end{array}
$$

From the induction hypothesis we get a plan P with length k that solves $\mathscr{P}'$. From (1) and (2), it follows that $\langle\, \mathsf{a}\,;\, \mathsf{P}\,\rangle$, with length $k+1$, solves $\mathscr{P}$. □

COROLLARY 8.28. *Let $\mathscr{P}$ be a planning problem and $\mathcal{P}$ be the cpps for $\mathscr{P}$. The following are equivalent:*

i. There is a plan P that solves $\mathscr{P}$.
ii. There is a proof of $\mathcal{P}$ in system ELS of the form given in Corollary 8.25.
iii. There is a proof of $\mathcal{P}$ that is constructed by applying the rule action inductively bottom-up for the action structures for the actions in P with respect to their order in P; and then finally by applying the rule $\mathsf{termination}$ when the plan is empty.

REMARK 8.29. *In the definition of cpps, where a conjunctive planning problem is encoded as an ELS structure, the exponential '! ' preceding the problem structure can be safely removed. However, in this case proofs of such structures are constructed without any instance of the rule $\mathsf{p}\downarrow$.*

REMARK 8.30. *The definition of a plan that solves a planning problem, which was used so far in this section, is too restrictive: The condition of this definition imposes the state that is reached by the plan, to be exactly the same as the goal state. However it is also feasible to replace the equality in the condition with multiset inclusion. In other words, the states that contain the resources of the goal state together with other resources can be considered as accepting states. Such a view of planning problems can be easily accommodated into the current definition by means of additional actions for consuming the excessive resources at the end of the execution of the plan: For each resource $r \in \mathscr{R}$, one can define an action that has only this resource as the condition and an empty effect.*

Let us see this on the following example.

EXAMPLE 8.31. *As before, Peter has a euro which he can change for two fifty cents. However, this time he is thirsty, so he only wants to get a lemonade. The actions for changing the euro and buying a lemonade are defined as before. However, now we are also equipped with the auxiliary action $\dot{\{}\, f \,\dot{\}} \to \dot{\emptyset}$ for getting rid of the "excessive" fifty cents. We get the following cpps for this planning problem*

$$[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, \bot), !\,[e, \bar{l}]\,]$$

which can be proved as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[?\,(\bar{e},[f,f]),?\,(\bar{f},l),?\,(\bar{f},\bot),!\,[l,\bar{l}]\,]}\,\mathsf{termination}}{[?\,(\bar{e},[f,f]),?\,(\bar{f},l),?\,(\bar{f},\bot),!\,[l,f,\bar{l}]\,]}\,\mathsf{action}}{[?\,(\bar{e},[f,f]),?\,(\bar{f},l),?\,(\bar{f},\bot),!\,[f,f,\bar{l}]\,]}\,\mathsf{action}}{[?\,(\bar{e},[f,f]),?\,(\bar{f},l),?\,(\bar{f},\bot),!\,[e,\bar{l}]\,]}\,\mathsf{action}$$

With Theorem 8.27, I provided a constructive proof of the equivalence of existence of a plan solving a planning problem and existence of a proof of the encoding of the planning problem in ELS. As we have seen in Corollary 8.28, the rules action and termination provide an algorithm for reading a plan from a proof that is constructed by using only these rules. However, because of the possible permutation of the inference rules, a proof of a cpps can be constructed in many different ways, and these instances do not provide an explicit reading of a plan from these different proofs.

In the following, I will give an algorithm for extracting partial order plans from the proofs of conjunctive planning problem structures. This way, I will establish an explicit correspondence between partial order plans and proofs of the cpps. Because of the explicit treatment of resources in conjunctive planning, these partial order plans respect a concurrency semantics, namely labelled event structure semantics.

8.3. Labelled Event Structure Semantics

Labelled event structures (LES) [**SNW96, WN95**] is a non-interleaving branching-time behavioral model of concurrency. An interleaving model of concurrency is equipped with an expansion law that identifies parallel composition by means of choice and sequential composition. In a nut shell, in an interleaving model, parallel composition of two events indicates that these events can take place in either order. A model for concurrency without such an expansion law is said to be a non-interleaving model: When two events are composed in parallel they can take place simultaneously or in either order. In such a view of the systems, the independence and causality between the events of the system is central. In a LES the causality between actions is captured in terms of their dependencies in a partial order.

In concurrency theory another discussion is centered around linear-time semantics versus branching-time semantics. In a linear-time semantics, two processes that agree on the ordering of actions are considered equivalent. However, such processes may differ in their branching structure. In this respect, the branching structure of a process is determined by the moments that choices between alternative branches of behaviour are made. A branching-time semantics distinguishes processes with the same ordering of actions but different branching structures.

Labelled event structures provide a branching-time semantics of the systems being modelled. Apart from the causality which is expressed in a partial order, in a LES, the nondeterminism in the computation is captured by a conflict relation, which is a symmetric irreflexive relation of events. In a planning perspective, this corresponds to actions that are applicable in the same state, but are in conflict. When two actions are in conflict with each other, execution of one of them instead of the other determines a different state space ahead. This provides a branching-time model of the possible computations.

In this section, I associate to every planning problem a LES that represents the independence and causality of all the actions performable in different states of a search for a plan. By resorting to the inference rule action, which gives the operational semantics of the conjunctive planning problems, I associate a transition system to each proof of a planning problem. I then adapt some ideas from [**Gug96**] where LES semantics for a class of linear logic proofs has been studied: I apply the techniques presented in [**Gug96**] to conjunctive planning problems to obtain LES from the transition systems. Following this, by relying on the notion of independence among actions provided by the explicit handling of the resources, I provide an algorithm to extract partial order plans that respects the LES semantics of the plans from the proofs of the planning problems.

8.3.1. From Proofs to Transition Systems. In order to obtain a characterization of the conjunctive planning problems that takes into account all the possible computations, I will first associate to each cpps a transition system. Such an explicit representation of the states is called a *system* model in [**SNW96, WN95**], in contrast to *behavioral* models, which abstract away from such information, and focus instead on the behavior in terms of patterns of occurrences of actions over time. The LES that will be obtained later is such a behavioral model.

Let us first recall the notion of a transition system.

DEFINITION 8.32. *A* transition system *is a 4-tuple* $\langle \mathcal{S}, s_I, \mathcal{L}, \rightarrow \rangle$ *where*

(1) $\mathcal{S}$ *is a set of* states*;*
(2) $s_I \in \mathcal{S}$ *is the initial state;*
(3) $\mathcal{L}$ *is a set of* labels*;*
(4) $\rightarrow \subseteq \mathcal{S}^2 \times \mathcal{L}$ *is the* labelled transition relation.

If $(s, s', \mathsf{a}) \in \rightarrow$ *we write* $s \xrightarrow{\mathsf{a}} s'$*. If* $s_0 \xrightarrow{\mathsf{a_1}} \ldots \xrightarrow{\mathsf{a_h}} s_h$*,* $h \geq 1$*, we write* $s_0 \overset{\mathsf{P}}{\twoheadrightarrow} s_h$*, where* $\mathsf{P} = \langle \mathsf{a_1}; \ldots; \mathsf{a_h} \rangle$ *or* $\mathsf{P} = \circ$*. Let* $s \xrightarrow{\circ} s$ *denote the empty composition of transitions. A state* s *is* reachable*, if* $s_I \overset{\mathsf{P}}{\twoheadrightarrow} s$ *for some* P*. A transition system is* reachable*, if every state in* $\mathcal{S}$ *is reachable. A transition system is* acyclic *if* $s \overset{\mathsf{P}}{\twoheadrightarrow} s$ *implies* $\mathsf{P} = \circ$*. Transitions are denoted by* t*. A sequence* $\langle t_1; t_2; \ldots \rangle$ *of transitions such that* $t_i = (s_{i-1}, s_i, \mathsf{a_i})$ *for* $i = 1, 2, \ldots$ *and* $s_0 = s_I$ *is called a* path*. A finite path* $\tau = \langle t_1; \ldots; t_h \rangle$ yields s_h*, if* $t_h = (s_{h-1}, s_h, \mathsf{a_h})$*. A path can also be denoted as* $s_0 \xrightarrow{\mathsf{a_1}} \ldots \xrightarrow{\mathsf{a_h}} s_h$*. The* length of a path *is the number of transitions in it.*

With the following definitions I will carry the notion of a transition system to the derivations where the notion of a derivation unifies with the notion of a state. States are derivations. The premises and conclusions of these derivations are cpps. Because a cpps $\mathcal{P}$ is also a derivation, it is also a state. The computation consists of moving from one state to another state. Labels are adopted to keep track of the actions that are selected and used at the application of the rule action.

DEFINITION 8.33. *Given a conjunctive planning problem* $\mathscr{P}$*, let* $\mathcal{P}$ *be the cpps for* $\mathscr{P}$*.* $\mathsf{TS}[\![\mathcal{P}]\!] = (\mathcal{S}, s_I, \mathscr{A}, \rightarrow)$ *is the reachable transition system such that* $(\Delta, \Delta', \mathsf{a}) \in \rightarrow$ *where* $\Delta, \Delta' \in \mathcal{S}$ *if and only if*

(*i*) $s_I = \mathcal{P}$*;*

(*ii*) *for some* $\mathcal{P}'$*,* Δ *has the shape* $\begin{matrix} \mathcal{P}' \\ \Delta \,[\!]\{\,\mathsf{action}\,\} \\ \mathcal{P} \end{matrix}$;

(*iii*) *for some* $\mathcal{P}''$*, there exists a derivation* $\mathsf{action}\,\dfrac{\mathcal{P}''}{\mathcal{P}'}$ *where the conjunctive action structure for the action* $\mathsf{a} \in \mathscr{A}$ *is used;*

(*iv*) Δ' *is the derivation*

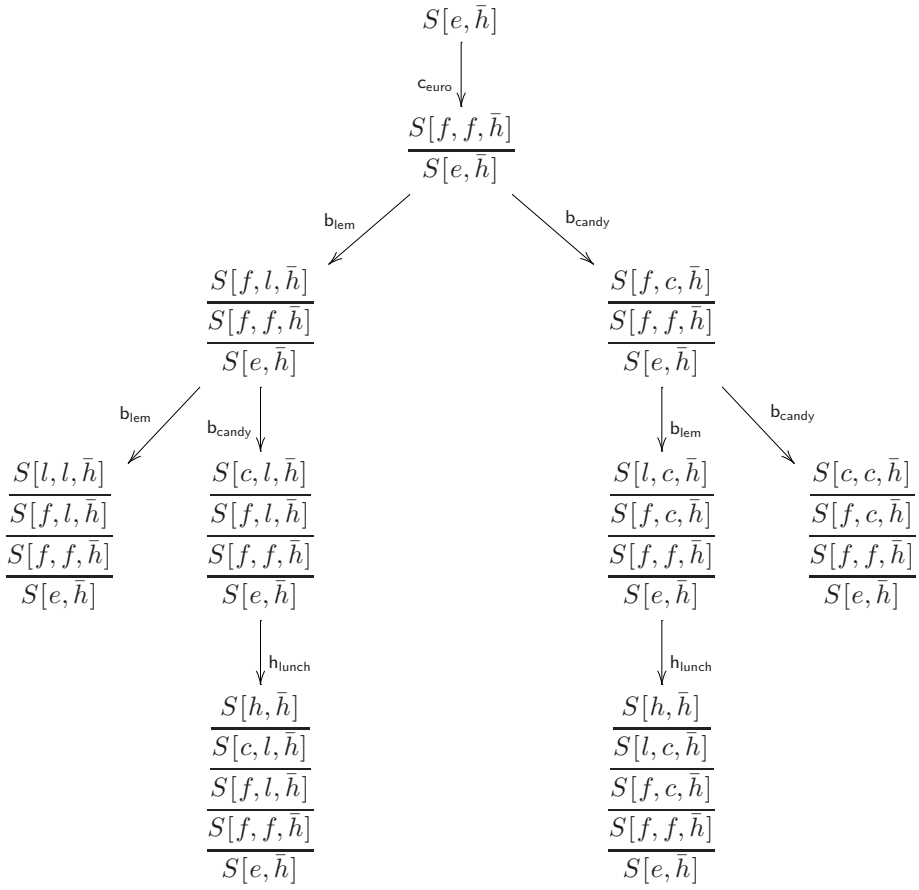
$$\mathsf{action}\,\frac{\mathcal{P}''}{\mathcal{P}'}$$
$$\Delta \Big\| \{\,\mathsf{action}\,\}$$
$$\mathcal{P} \qquad .$$

We then write $\Delta \xrightarrow{\mathsf{a}} \Delta'$.

Example 8.34. *Let us consider the cpps* $\mathcal{P}$ *for the planning problem of Example 8.5. Let* $S\{\ \}$ *denote the structure context*

$$[?\,(\bar{e},[f,f]),?\,(\bar{f},l),?\,(\bar{f},c),?\,(\bar{l},\bar{c},h),!\{\ \}]\ .$$

The transition system $\mathsf{TS}[\![\mathcal{P}\,]\!]$ *is as follows:*



Proposition 8.35. *Given a cpps* $\mathcal{P}$*,* $\mathsf{TS}[\![\mathcal{P}\,]\!]$ *is acyclic;*

PROOF. Every transition transforms a derivation into a syntactically bigger derivation. □

DEFINITION 8.36. *Let $\mathcal{P}$ be a cpps and $\tau = \langle t_1; \ldots; t_h \rangle$ be a finite path in $\mathsf{TS}[\![\mathcal{P}]\!]$. τ is* a path yielding Δ_h, *if, for $1 \leq i \leq h$, we have that $t_i = (\Delta_{i-1}, \Delta_i, \mathsf{a}_i)$. If the premise of Δ_h is $\mathcal{P}'$ and the rule* termination *is applicable to $\mathcal{P}'$, then τ is* a successful path. *If the rule* termination *is not applicable to $\mathcal{P}'$ and there is no Δ in $\mathsf{TS}[\![\mathcal{P}]\!]$ such that for any $\mathsf{a} \in \mathscr{A}$, $\Delta_h \xrightarrow{\mathsf{a}} \Delta$, then τ is called* a failed path.

EXAMPLE 8.37. *Consider the transition system $\mathsf{TS}[\![\mathcal{P}]\!]$ of Example 8.34. The following is a successful path because the rule* termination *can be applied to the last derivation of the path:*

$$S[e,\bar{h}] \overset{\mathsf{c_{euro}}}{\longrightarrow} \frac{S[f,f,\bar{h}]}{S[e,\bar{h}]} \overset{\mathsf{b_{lem}}}{\longrightarrow} \frac{\dfrac{S[f,l,\bar{h}]}{S[f,f,\bar{h}]}}{S[e,\bar{h}]} \overset{\mathsf{b_{candy}}}{\longrightarrow} \frac{\dfrac{\dfrac{S[c,l,\bar{h}]}{S[f,l,\bar{h}]}}{S[f,f,\bar{h}]}}{S[e,\bar{h}]} \overset{\mathsf{h_{lunch}}}{\longrightarrow} \frac{\dfrac{\dfrac{\dfrac{S[h,\bar{h}]}{S[c,l,\bar{h}]}}{S[f,l,\bar{h}]}}{S[f,f,\bar{h}]}}{S[e,\bar{h}]}$$

The following is a failed path because neither the rule termination *can be applied to the last derivation of the path, nor is there a possible transition from this derivation.*

$$S[e,\bar{h}] \overset{\mathsf{c_{euro}}}{\longrightarrow} \frac{S[f,f,\bar{h}]}{S[e,\bar{h}]} \overset{\mathsf{b_{lem}}}{\longrightarrow} \frac{\dfrac{S[f,l,\bar{h}]}{S[f,f,\bar{h}]}}{S[e,\bar{h}]} \overset{\mathsf{b_{lem}}}{\longrightarrow} \frac{\dfrac{\dfrac{S[l,l,\bar{h}]}{S[f,l,\bar{h}]}}{S[f,f,\bar{h}]}}{S[e,\bar{h}]}$$

PROPOSITION 8.38. *There is a proof of a cpps $\mathcal{P}$ in* ELS *with k number of applications of the rule $\mathsf{p}\!\downarrow$ if and only if there is a successful path in $\mathsf{TS}[\![\mathcal{P}]\!]$ with length k.*

PROOF. From Corollary 8.28, it follows that a plan P with length k solves a planning problem if and only if there is a successful path in $\mathsf{TS}[\![\mathcal{P}]\!]$ with length k. The result follows from Theorem 8.27, because there is a plan with length k that solves the planning problem if and only if there is a proof of a cpps $\mathcal{P}$ with k number of applications of the rule $\mathsf{p}\!\downarrow$. □

The transition system of Example 8.34 is a transition system with finite number of states. Because in planning one is usually interested in finite computations, I used this example so far. However from the point of view of concurrency theory, often infinite computations need to be modeled. For instance, if one considers modeling an operating system, ideally the computations should not terminate. In order to be able to argue about such infinite computations, I will now extend the example of this chapter in a way that accommodates an infinite computation:

EXAMPLE 8.39. *Consider the planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$ of Example 8.5 where we have the scenario where Peter is hungry and wants to have lunch with his euro. I now extend the planning problem $\mathscr{P}$ by allowing Peter to sell his candy-bar and lemonade to a colleague for a euro whenever he wants. Thus, the planning problem $\mathscr{P}' = \langle \mathscr{R}, \mathscr{A}', \mathcal{I}, \mathcal{G} \rangle$ is obtained from the planning problem $\mathscr{P}$ by extending the set $\mathscr{A}$ with the action $\mathsf{s_{lunch}}$ such that*

$$\mathscr{A}' = \mathscr{A} \cup \{\, \mathsf{s_{lunch}} : \dot{\{}\, l, c \,\dot{\}} \to \dot{\{}\, e \,\dot{\}} \,\}.$$

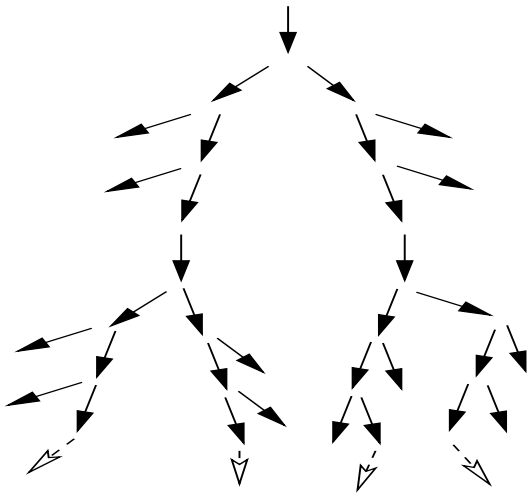


FIGURE 8.2. The transition system $\mathsf{TS}[\![\mathcal{P}']\!]$

Let $\mathcal{P}'$ be the cpps for $\mathscr{P}'$. This way, we can observe infinite computations in the transition system $\mathsf{TS}[\![\mathcal{P}']\!]$, which is depicted in Figure 8.2. In this figure, each arrow denotes a transition in $\mathsf{TS}[\![\mathcal{P}']\!]$.

In a first step towards observing the independence and the causality in the derivations, we will now consider two derivations equivalent if they have the same premise and conclusion. The following definition serves this purpose:

DEFINITION 8.40. *Let $\mathcal{D}$ be the set of derivations, and $\mathcal{P}$ and $\mathcal{P}'$ be cpps. $\approx \subset \mathcal{D}^2$ is the least equivalence relation such that $\Delta \approx \Delta'$ if and only if*

$$\begin{array}{c}\mathcal{P}'\\ \Delta\big\|\mathsf{ELS}\\ \mathcal{P}\end{array} \quad \textit{and} \quad \begin{array}{c}\mathcal{P}'\\ \Delta'\big\|\mathsf{ELS}\\ \mathcal{P}\end{array} .$$

$[\Delta]_\approx$ denotes the equivalence class of the derivation Δ under $\approx$. The set $\mathcal{D}/\approx$, the set of equivalence classes of derivations under $\approx$, is called the set of abstract derivations. *The elements of $\mathcal{D}/\approx$ are denoted by δ.*

EXAMPLE 8.41. *Let us consider the cpps of Example 8.34. We have the following syntactically different derivations that are equivalent under $\approx$:*

$$\dfrac{\dfrac{\dfrac{S[l,c,\bar{h}]}{S[f,c,\bar{h}]}}{S[f,f,\bar{h}]}}{S[e,\bar{h}]} \quad \approx \quad \dfrac{\dfrac{\dfrac{S[l,c,\bar{h}]}{S[l,f,\bar{h}]}}{S[f,f,\bar{h}]}}{S[e,\bar{h}]}$$

PROPOSITION 8.42. *If two states of $\mathsf{TS}[\![\mathcal{P}]\!]$, Δ and Δ', are equivalent under $\approx$ and if $\Delta \xrightarrow{\mathsf{a}} \Delta''$, then, in $\mathsf{TS}[\![\mathcal{P}]\!]$, there is a transition $\Delta' \xrightarrow{\mathsf{a}} \Delta'''$ such that $\Delta'' \approx \Delta'''$.*

PROOF. Because Δ'' and Δ''' have the same premises, same inference rules can be applied to the premises of these two derivations. □

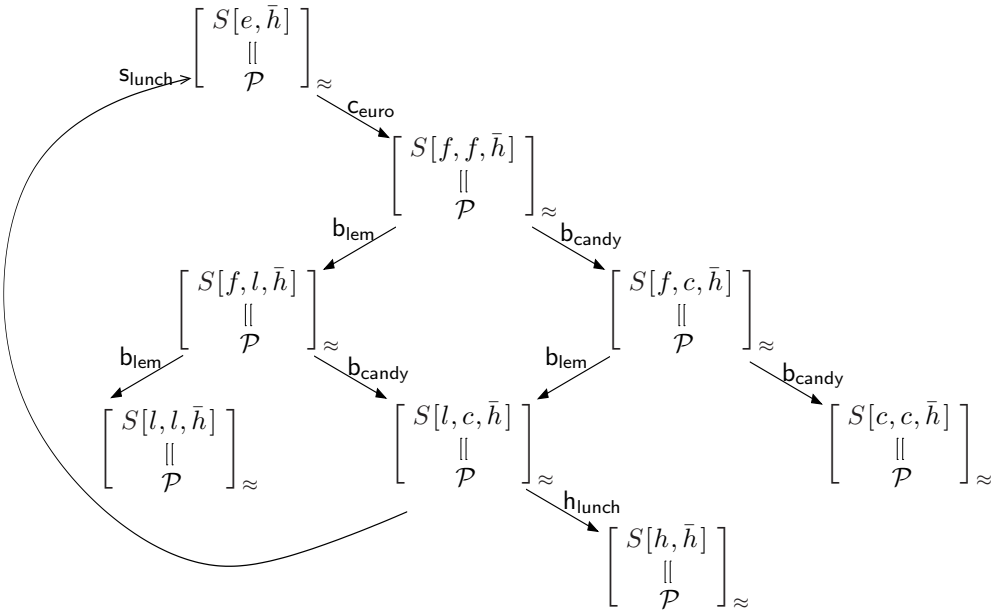


FIGURE 8.3. A transition system $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$

With the following definition, we will see a new transition system associated with a cpps that respects the equivalence of derivations induced by the relation $\approx$.

DEFINITION 8.43. *Given a cpps $\mathcal{P}$ and a $\mathsf{TS}[\![\mathcal{P}]\!] = (\mathcal{S}, s_I, \mathscr{A}, \rightarrow)$, let $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!] = (\mathcal{S}_{\approx}, s_{I\approx}, \mathscr{A}, \rightarrow_{\approx})$ be the transition system such that*

(i) $s_{I\approx} = \mathcal{P}$;
(ii) $\mathcal{S}_{\approx} = \mathcal{S}/\approx$;
(iii) $[\Delta]_{\approx} \xrightarrow{\mathsf{a}} [\Delta']_{\approx}$ *if and only if* $\Delta \xrightarrow{\mathsf{a}} \Delta'$ *where* $\mathsf{a} \in \mathscr{A}$.

EXAMPLE 8.44. *Let $S\{\ \}$ denote the structure context*

$$[?\,(\bar{e}, [f, f]), ?\,(\bar{f}, l), ?\,(\bar{f}, c), ?\,(\bar{l}, \bar{c}, h), ?\,(\bar{l}, \bar{c}, e), !\{\ \}]\ .$$

The transition system $\mathsf{TS}_{\approx}[\![\mathcal{P}']\!]$ for the cpps $\mathcal{P}'$ of Example 8.39 is in Figure 8.3.

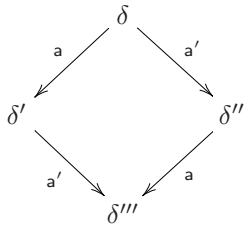
DEFINITION 8.45. *Let $\mathcal{P}$ be a cpps and $\tau = \langle t_1; \ldots; t_h \rangle$ be a finite path in $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$. τ is called an* abstract path yielding δ_h*, if, for all $1 \leq i \leq h$, $t_i = (\delta_{i-1}, \delta_i, a_i)$. If $\mathcal{P}'$ is the premise of (all the elements of) δ_h and the rule* termination *is applicable to $\mathcal{P}'$, then τ is a* successful abstract path*. If the rule* termination *is not applicable to $\mathcal{P}'$ and there is no δ in $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ such that, for any a, $\delta_h \xrightarrow{\mathsf{a}} \delta$ then τ is a* failed abstract path.

PROPOSITION 8.46. *Given a cpps $\mathcal{P}$, $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ is reachable.*

PROOF. Because $\mathsf{TS}[\![\mathcal{P}]\!]$ is by definition reachable, $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ is also reachable. □

The intuition behind the following definition is to capture the independence and causality between actions. Informally, two actions a and a' are independent if their ordering does not influence the reachability of a certain state that is common to both paths.

DEFINITION 8.47. *Given a cpps $\mathcal{P}$ and $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$, let the relation $\diamondsuit \subseteq \mathscr{A}^2 \times \mathcal{S}_{\approx}^4$ be such that $(\mathsf{a}, \mathsf{a}', \delta, \delta', \delta'', \delta''') \in \diamondsuit$ if and only if $(\delta, \delta', \mathsf{a})$, $(\delta, \delta'', \mathsf{a}')$, $(\delta', \delta''', \mathsf{a}')$, and $(\delta'', \delta''', \mathsf{a})$ are transitions in $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ where $(\delta, \delta', \mathsf{a}) \neq (\delta, \delta'', \mathsf{a}')$:*



We will call $\diamondsuit$ the diamond property *of $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$.*

EXAMPLE 8.48. *Considering the cpps $\mathcal{P}$ of our running example, let*

$$\delta = \left[\begin{array}{c} S[f, f, \bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},$$

$$\delta' = \left[\begin{array}{c} S[l, f, \bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx}, \qquad \delta'' = \left[\begin{array}{c} S[f, c, \bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},$$

$$\delta''' = \left[\begin{array}{c} S[l, c, \bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx};$$

and let $\mathsf{a} = \mathsf{b}_{\mathsf{lem}}$ and $\mathsf{a}' = \mathsf{b}_{\mathsf{candy}}$. Then we have $(\mathsf{a}, \mathsf{a}', \delta, \delta', \delta'', \delta''') \in \diamondsuit$.

DEFINITION 8.49. *Given $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!] = (\mathcal{S}_{\approx}, \mathcal{P}, \mathscr{A}, \rightarrow_{\approx})$ and its diamond property $\diamondsuit$, the relation $\backsimeq$ is the least equivalence relation on its paths such that the following holds: given two paths of the form*

$$\tau_1 = \langle\, t\,;\, (\delta, \delta', \mathsf{a})\,;\, (\delta', \delta''', \mathsf{a}')\,;\, t' \rangle \quad \text{and} \quad \tau_2 = \langle t\,;\, (\delta, \delta'', \mathsf{a}')\,;\, (\delta'', \delta''', \mathsf{a})\,;\, t' \rangle\,,$$

if $(\mathsf{a}, \mathsf{a}', \delta, \delta', \delta'', \delta''') \in \diamondsuit$, then $\tau_1 \backsimeq \tau_2$.

The following proposition captures the intuition of the above definitions with respect to equivalent paths in a $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$.

PROPOSITION 8.50. *Given finite paths τ_1 and τ_2 in $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$, if $\tau_1 \backsimeq \tau_2$, then they both yield the same state.*

PROOF. Follows immediately from Definition 8.47 and Definition 8.49. □

8.3.2. Labelled Event Structures of Planning Problems. In this section, we will see how we can associate to every cpps a *labelled event structure*. In the current setting, events correspond to certain instances of actions. In a LES events are partially ordered and there is a conflict relation among the events. This conflict relation represents the nondeterminism in the system. Events that are not in a conflict can be freely executed in a way which respects the order determined by the partial order. Labelled event structures provide a clear computational model which captures the concurrent behavior of events while respecting their independence and causality.

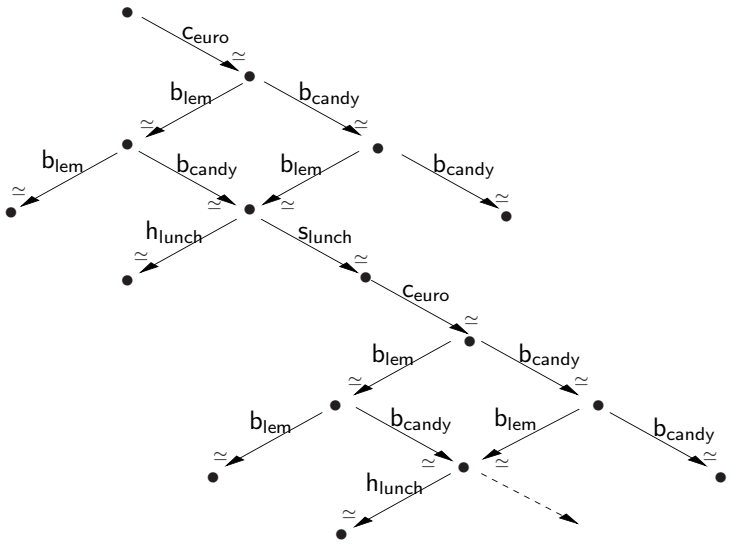


FIGURE 8.4. The transition system $\mathsf{TS}_{\simeq}[\![\mathcal{P}']\!]$

In the following, the labelled event structure for a cpps $\mathcal{P}$ will be obtained from the transition system $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ and the equivalence relation $\backsimeq$ on its paths. For this purpose, I will first define a new transition system, such that in these transition systems all paths reaching a certain state belong to the same equivalence class induced by $\backsimeq$. A general exposure of these results for a class of linear logic proofs can be found in [**Gug96**]. For an exposure of the relationship between transition systems, labelled event structures, and some other models for concurrency the reader is referred to [**SNW96, WN95**].

DEFINITION 8.51. *Given a cpps $\mathcal{P}$ and $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!] = (\mathcal{S}_{\approx}, s_{I\approx}, \mathscr{A}, \rightarrow_{\approx})$, let $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!] = (\mathcal{S}_{\simeq}, s_{I\simeq}, \mathscr{A}, \rightarrow_{\simeq})$ be the transition system such that*

(*i*) *$\mathcal{S}_{\simeq} = \mathcal{T}/\simeq$, where $\mathcal{T}$ is the set of finite paths in $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ and $\simeq$ is the equivalence relation on its paths induced by the diamond property $\diamondsuit$ of $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$. Elements of $\mathcal{S}_{\simeq}$ are denoted by π;*

(*ii*) *$s_{I\simeq} = [\circ]_{\simeq}$;*

(*iii*) *$[\tau]_{\simeq} \xrightarrow{\mathsf{a}}_{\simeq} [\tau']_{\simeq}$ if and only if $\tau' \simeq \langle\, \tau\,;\, (\delta, \delta', \mathsf{a})\,\rangle$ where $(\delta, \delta', \mathsf{a}) \in \rightarrow_{\approx}$.*

EXAMPLE 8.52. *The transition system $\mathsf{TS}_{\simeq}[\![\mathcal{P}']\!]$ for the running example is shown in Figure 8.4.*

PROPOSITION 8.53. *For every cpps $\mathcal{P}$, $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$ is reachable and acyclic.*

PROOF. $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$ is obtained from $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ which is reachable. Because each transition transforms an abstract path to a syntactically bigger abstract path, $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$ is acyclic. □

DEFINITION 8.54. *A* labelled event structure *is a structure $(E, \leq, \#, \mathcal{L}, \ell)$, where*

(*i*) E *is a set of events;*

(*ii*) $\leq \subseteq E^2$ *is a partial order such that for every* $\mathsf{e} \in E$ *the set* $\{\mathsf{e}' \in E \mid \mathsf{e}' \leq \mathsf{e}\}$ *is finite;*

(*iii*) *the* conflict relation $\# \subseteq E^2$ *is a symmetric and irreflexive relation such that if* $\mathsf{e} \# \mathsf{e}'$ *and* $\mathsf{e}' \leq \mathsf{e}''$, *then* $\mathsf{e} \# \mathsf{e}''$, *for every* $\mathsf{e}, \mathsf{e}', \mathsf{e}'' \in E$;

(*iv*) $\mathcal{L}$ *is a set of labels;*

(*v*) $\ell : E \to \mathcal{L}$ *is a labeling function.*

A LES of a cpps $\mathcal{P}$ is obtained from $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$. For this purpose let me first lift the diamond property from $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$ to $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$, that is, from the equivalence classes of derivations to the equivalence classes of paths:

Definition 8.55. *Given* $\mathcal{P}$ *and the diamond property* $\diamondsuit$ *of* $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$, *we define* $\diamondsuit_{\simeq} \subset \mathscr{A}^2 \times \mathcal{S}^4_{\simeq}$ *for* $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!]$ *as follows: For some abstract paths* τ, τ', τ'' *and* τ''',

$$(\mathsf{a}, \mathsf{a}', [\tau]_{\simeq}, [\tau']_{\simeq}, [\tau'']_{\simeq}, [\tau''']_{\simeq}) \in \diamondsuit_{\simeq} \quad \textit{if and only if}$$

$$\tau' \simeq \langle\, \tau\,; (\delta, \delta', \mathsf{a})\,\rangle\,, \quad \tau'' \simeq \langle\, \tau\,; (\delta, \delta'', \mathsf{a}')\,\rangle\,, \quad \tau''' \simeq \langle\, \tau'\,; (\delta', \delta''', \mathsf{a}')\,\rangle\,,$$

$$\tau''' \simeq \langle\, \tau''\,; (\delta'', \delta''', \mathsf{a})\,\rangle \;\; \textit{and} \;\; (\mathsf{a}, \mathsf{a}', \delta, \delta', \delta'', \delta''') \in \diamondsuit$$

for some states $\delta, \delta', \delta''$ *and* δ''' *of* $\mathsf{TS}_{\approx}[\![\mathcal{P}]\!]$.

Example 8.56. *Consider the* $\mathsf{a}, \mathsf{a}', \delta, \delta', \delta''$ *and* δ''' *of Example 8.48. Let* τ *be an abstract path that leads to a derivation* Δ *with the structure* $S[f, f, \bar{h}]$ *at the premise. Then we have*

$$\tau' \simeq \left\langle\ \tau\ ;\ \left(\ \left[\begin{array}{c} S[f,f,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \left[\begin{array}{c} S[l,f,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \mathsf{b}_{\mathsf{lem}}\ \right)\ \right\rangle\ ,$$

$$\tau'' \simeq \left\langle\ \tau\ ;\ \left(\ \left[\begin{array}{c} S[f,f,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \left[\begin{array}{c} S[f,c,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \mathsf{b}_{\mathsf{candy}}\ \right)\ \right\rangle\ ,$$

$$\tau''' \simeq \left\langle\ \tau'\ ;\ \left(\ \left[\begin{array}{c} S[l,f,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \left[\begin{array}{c} S[l,c,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \mathsf{b}_{\mathsf{candy}}\ \right)\ \right\rangle\ ,$$

$$\tau''' \simeq \left\langle\ \tau''\ ;\ \left(\ \left[\begin{array}{c} S[f,c,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \left[\begin{array}{c} S[l,c,\bar{h}] \\ \| \\ \mathcal{P} \end{array}\right]_{\approx},\ \mathsf{b}_{\mathsf{lem}}\ \right)\ \right\rangle\ .$$

Thus, we have $(\mathsf{b}_{\mathsf{lem}}, \mathsf{b}_{\mathsf{candy}}, [\tau]_{\simeq}, [\tau']_{\simeq}, [\tau'']_{\simeq}, [\tau''']_{\simeq}) \in \diamondsuit_{\simeq}$

Definition 8.57. *Given* $\mathsf{TS}_{\simeq}[\![\mathcal{P}]\!] = (\mathcal{S}_{\simeq}, \mathcal{P}, \mathscr{A}, \to_{\simeq})$ *and its diamond property* $\diamondsuit_{\simeq}$, *let* $\sim$ *be the least equivalence relation on* $t, t' \in \to_{\simeq}$ *such that*

$$t \sim t' \;\; \textit{if and only if} \;\; t = (\pi, \pi', \mathsf{a})\,,\ t' = (\pi'', \pi''', \mathsf{a})$$

and there exists $\mathsf{a}' \in \mathscr{A}$ *such that* $(\mathsf{a}, \mathsf{a}', \pi, \pi', \pi'', \pi''') \in \diamondsuit_{\simeq}$.

Intuitively, two transitions are in $\sim$ if they represent the same event. Let us see this on an example:

EXAMPLE 8.58. *Consider the relation* $\diamondsuit_\simeq$ *of Example 8.56 where we have*

$$(\mathsf{b}_{\mathsf{lem}}, \mathsf{b}_{\mathsf{candy}}, [\tau]_\simeq, [\tau']_\simeq, [\tau'']_\simeq, [\tau''']_\simeq) \in \diamondsuit_\simeq.$$

If $\pi = [\tau]_\simeq$, $\pi' = [\tau']_\simeq$, $\pi'' = [\tau'']_\simeq$ *and* $\pi''' = [\tau''']_\simeq$, *as in Definition 8.57, then we have* $t \sim t'$ *where*

$$t = \left(\left[\begin{array}{c} S[f, f, \bar{h}] \\ \| \\ \mathcal{P} \end{array} \right]_\approx , \left[\begin{array}{c} S[l, f, \bar{h}] \\ \| \\ \mathcal{P} \end{array} \right]_\approx , \mathsf{b}_{\mathsf{lem}} \right) \quad \textit{and}$$

$$t' = \left(\left[\begin{array}{c} S[l, f, \bar{h}] \\ \| \\ \mathcal{P} \end{array} \right]_\approx , \left[\begin{array}{c} S[l, c, \bar{h}] \\ \| \\ \mathcal{P} \end{array} \right]_\approx , \mathsf{b}_{\mathsf{lem}} \right) .$$

DEFINITION 8.59. *Given a cpps* $\mathcal{P}$ *and* $\mathsf{TS}_\simeq[\![\mathcal{P}]\!] = (\mathcal{S}_\simeq, \mathcal{P}, \mathscr{A}, \ell)$, *let* $\mathsf{LES}[\![\mathcal{P}]\!] = (E, \leq, \#, \mathscr{A}, \ell)$ *be the labelled event structure such that*

(*i*) $E = \rightarrow_\simeq / \sim$;

(*ii*) $\leq$ *is the reflexive closure of* $<$, *which is defined as follows: for all* $\mathsf{e}, \mathsf{e}' \in E$, $\mathsf{e} < \mathsf{e}'$ *if and only if* $\mathsf{e} = [t]_\sim$ *and* $\mathsf{e}' = [t']_\sim$, *and for every path* τ *in* $\mathsf{TS}_\simeq[\![\mathcal{P}]\!]$ *and for every* $t''' \in \rightarrow_\simeq$ *such that* $\langle \tau \,;\, t''' \rangle$ *is a path and* $t''' \sim t'$, *there exists* $t'' \sim t$ *such that* $\tau = \langle \tau' ; t'' ; \tau'' \rangle$ *for some* τ', τ'' ;

(*iii*) $[t]_\sim \# [t']_\sim$ *if and only if for every path* τ *in* $\mathsf{TS}_\simeq[\![\mathcal{P}]\!]$ *and for every* $t'', t''' \in \rightarrow_\simeq$ *such that* $t \sim t''$ *and* $t' \sim t'''$, *if* t'' *appears in* τ, *then* t''' *does not appear in* τ;

(*iv*) $\ell([(\pi, \pi', \mathsf{a})]_\sim) = \mathsf{a}$.

EXAMPLE 8.60. *The labelled event structure* $\mathsf{LES}[\![\mathcal{P}']\!]$ *for the cpps* $\mathcal{P}'$ *of Example 8.39 is as in Figure 8.5. The nodes of the graph are delivered by the function* ℓ *such that* $\ell([(\pi, \pi', \mathsf{a})]_\sim) = \mathsf{a}$

The relation $\leq$ of Definition 8.59 is a partial order relation which provides a representation of independence and causality between different events. The events that are not ordered with respect to $\leq$ are independent, thus they can co-occur. The events that are ordered follow a chain of causality, that is, for an event e, all events $\mathsf{e}' < \mathsf{e}$, the execution of e is impossible without the prior execution of e'. Thus, the relation $\leq$ succeeds in presenting the independence and causality between events when different actions are concerned. However, due to the condition $(\delta, \delta', \mathsf{a}) \neq (\delta, \delta'', \mathsf{a}')$ in Definition 8.47, when identical actions are concerned we can not observe their independence in the relation $\leq$. In other words, with the above definitions, it is impossible to observe parallelism whenever two identical actions are applied to two different branches of a derivation which both have the same premise.

EXAMPLE 8.61. *Consider the* $\mathsf{LES}[\![\mathcal{P}']\!]$ *of Example 8.60. For some transitions* $\delta \neq \delta' \neq \delta''$, *we have that* $(\delta, \delta', \mathsf{b}_{\mathsf{lem}}) < (\delta', \delta'', \mathsf{b}_{\mathsf{lem}})$ *although these two actions can be executed in parallel.*

The following definition modifies the relation $\leq$ in a way that allows to observe a parallelism such that whenever there is a transition $\delta \xrightarrow{\mathsf{a}} \delta' \xrightarrow{\mathsf{a}} \delta''$ in $\mathsf{TS}[\![\mathcal{P}]\!]$ then the two actions involved may be exchanged.

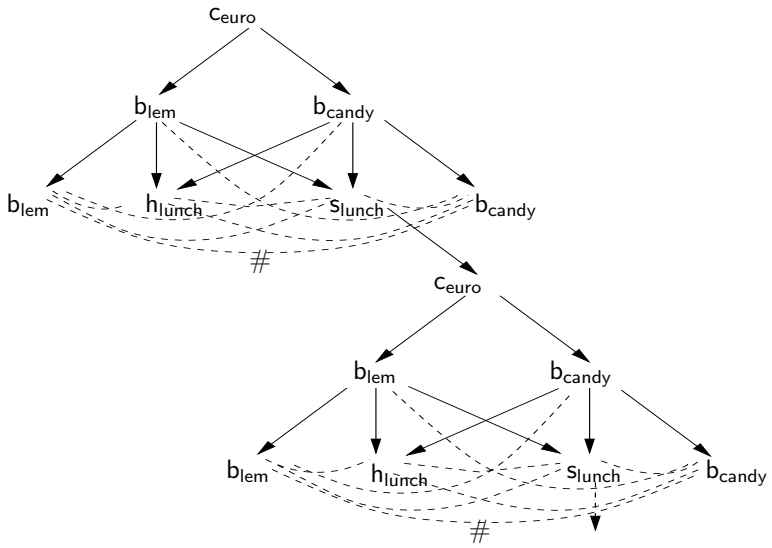


FIGURE 8.5. The labelled event structure $\mathsf{LES}[\![\mathcal{P}']\!]$

DEFINITION 8.62. *Given* $\mathsf{LES}[\![\mathcal{P}]\!] = (E, \leq, \#, \mathscr{A}, \ell)$*, for an event* $\mathsf{e} \in E$*, let*

$$\|\,\mathsf{e}\,\| = \{\mathsf{e}' \in E \mid \mathsf{e}' < \mathsf{e}, \forall \mathsf{e}'' \in E : (\,\mathsf{e}' \leq \mathsf{e}'' < \mathsf{e} \;\Rightarrow\; \ell(\mathsf{e}'') = \ell(\mathsf{e})\,)\,\}\,.$$

Then the relation $\leq^\star$ *is defined as follows:*

$$\leq^\star \;=\; \leq \;\setminus\; \bigcup_{\mathsf{e} \in E} \{\,(\mathsf{e}', \mathsf{e}) \mid \mathsf{e}' \in \|\,\mathsf{e}\,\|\,\}$$

By using the relation $\leq^\star$, we obtain a new LES:

DEFINITION 8.63. *Given a cpps* $\mathcal{P}$ *and* $\mathsf{LES}[\![\mathcal{P}]\!] = (E, \leq, \#, \mathscr{A}, \ell)$*, let*

$$\mathsf{LES}^\star[\![\mathcal{P}]\!] = (E, \leq^\star, \#, \mathscr{A}, \ell)\,.$$

EXAMPLE 8.64. *The labelled event structure* $\mathsf{LES}^\star[\![\mathcal{P}']\!]$ *for the cpps* $\mathcal{P}'$ *of Example 8.39 is as in Figure 8.6. It is important to observe that the two events with the actions* $\mathsf{b}_{\mathsf{lem}}$ *(*$\mathsf{b}_{\mathsf{candy}}$*), which were previously ordered in* $\mathsf{LES}[\![\mathcal{P}']\!]$*, are not ordered with respect to* $\leq^\star$ *in* $\mathsf{LES}^\star[\![\mathcal{P}']\!]$*.*

A labelled event structure of a conjunctive planning problem gives a concurrent model of all the possible executions of plans for this planning problem. With the definition below, I will give a formal characterization of these executions:

DEFINITION 8.65. *Given a* LES $(E, \leq, \#, \mathscr{A}, \ell)$*, for an event* $\mathsf{e} \in E$*,* $\lfloor \mathsf{e} \rfloor$ *denotes the set* $\{\mathsf{e}' \in E \mid \mathsf{e}' < \mathsf{e}\}$ *of* causes *of event* e*.*

DEFINITION 8.66. *Given a* LES $(E, \leq, \#, L, \ell)$*,* $\mathscr{C} \subseteq E$ *is a* configuration *if and only if*

(*i*) *for all* $\mathsf{e} \in \mathscr{C}$ *we have that* $\lfloor \mathsf{e} \rfloor \subset \mathscr{C}$ *;*

(*ii*) *for all* $\mathsf{e}, \mathsf{e}' \in \mathscr{C}$*, it is not the case that* $\mathsf{e} \# \mathsf{e}'$*.*

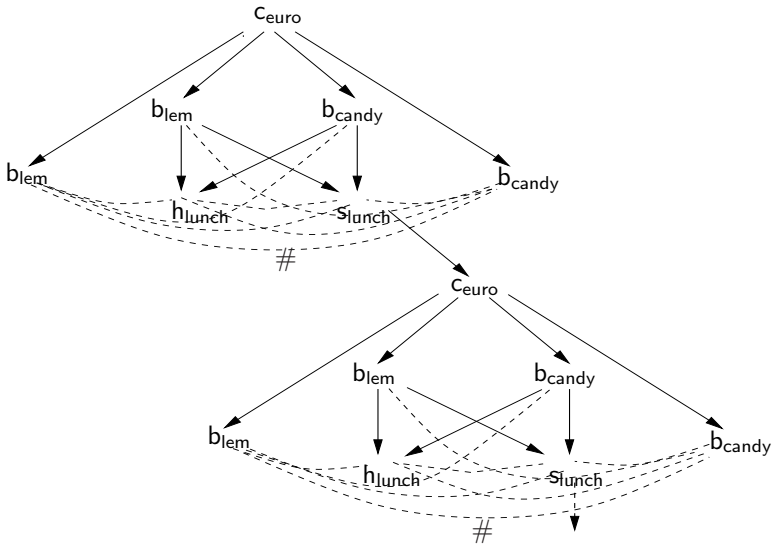


FIGURE 8.6. The labelled event structure $\mathsf{LES}^\star[\![\mathcal{P}']\!]$

DEFINITION 8.67. *Given a* LES $(E, \leq, \#, L, \ell)$*, and one of its configurations* $\mathscr{C}$*, we say that* event e is enabled at $\mathscr{C}$ *(denoted by* $\mathscr{C} \rhd \mathsf{e}$*) if and only if*

(*i*) $\mathsf{e} \notin \mathscr{C}$;

(*ii*) $\lfloor \mathsf{e} \rfloor \subseteq \mathscr{C}$;

(*iii*) $\mathsf{e}' \# \mathsf{e}$ *implies* $\mathsf{e}' \notin \mathscr{C}$.

DEFINITION 8.68. *Given a* LES $(E, \leq, \#, L, \ell)$ *and a finite sequence of events* $\mathsf{S} = \langle e_1; \ldots; e_h \rangle$, S *is a* securing for $\mathscr{C}$ *if and only if* $\mathscr{C} = \{e_1, \ldots, e_h\}$ *is a configuration and, for all* $1 \leq i \leq h$, $\{e_1, \ldots, e_{i-1}\} \rhd e_i$.

EXAMPLE 8.69. *Consider* $\mathsf{LES}^\star[\![\mathcal{P}']\!]$ *of Example 8.64 that is depicted in Figure 8.6. From the fragment of* $\mathsf{LES}^\star[\![\mathcal{P}']\!]$ *which is depicted on the right-hand side of Figure 8.7, we obtain a configuration* $\mathscr{C} = \{ \mathsf{c_{euro}}, \mathsf{b_{lem}}, \mathsf{b_{candy}} \}$. *We observe that* $\mathscr{C}$ *enables the event* $\mathsf{h_{lunch}}$*, i.e.,* $\mathscr{C} \rhd \mathsf{h_{lunch}}$. *We obtain the securing*

$$\langle \mathsf{c_{euro}}; \mathsf{b_{lem}}; \mathsf{b_{candy}}; \mathsf{h_{lunch}} \rangle$$

as it is depicted in Figure 8.7.

The following results are special cases of more general results in [**Gug96**]. Their proofs can be found in [**Gug96**]. They demonstrate the formal correspondence between the transition systems $\mathsf{TS}[\![\mathcal{P}]\!]$ and the $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ of a cpps $\mathcal{P}$.

THEOREM 8.70. *Given a cpps* $\mathcal{P}$, $\mathsf{LES}^\star[\![\mathcal{P}]\!] = (E, \leq^\star, \#, \mathscr{A}, \ell)$ *and a securing* S *in* $\mathsf{LES}^\star[\![\mathcal{P}]\!]$*, there is a path* $\mathcal{P} \stackrel{\ell(\mathsf{S})}{\twoheadrightarrow} \Delta$ *in* $\mathsf{TS}[\![\mathcal{P}]\!]$.

THEOREM 8.71. *Given a cpps* $\mathcal{P}$ *and a path* $\mathcal{P} \stackrel{\mathsf{a_1}}{\rightarrow} \Delta_1 \stackrel{\mathsf{a_2}}{\rightarrow} \cdots \stackrel{\mathsf{a_h}}{\rightarrow} \Delta_h$ *in* $\mathsf{TS}[\![\mathcal{P}]\!]$*, there is a securing* S *in* $\mathsf{LES}^\star[\![\mathcal{P}]\!] = (E, \leq^\star, \#, \mathscr{A}, \ell)$ *such that* $\ell(\mathsf{S}) = \langle \mathsf{a_1}; \ldots; \mathsf{a_h} \rangle$.

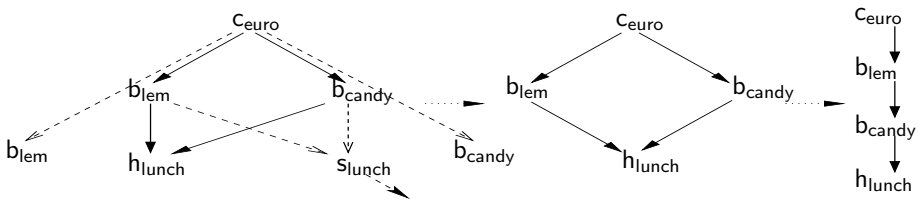


FIGURE 8.7. A securing obtained from $\mathsf{LES}^\star[\![\mathcal{P}']\!]$

8.3.3. Partial Order Plans, Plans, and Securings. A partial order plan with an LES semantics, which I presented in the previous subsection, can be extracted from the proof (or the derivation) of a conjunctive planning problem structure. This can be done by writing down constraints for atoms that get annihilated in a *par* structure by the application of the rule $\mathsf{ai}{\downarrow}$ during the proof construction. This is possible because the instances of this rule carries the information about the interactions, thus the dependencies, between actions. In the following, I will present an algorithm, i.e., recursive function that extracts this information. The intuition behind this algorithm is as follows: We mark each atom in an action structure with the name of that action. We also mark the atoms in the *problem structure*, i.e., the positive atoms in the *par structure* with the label init and the negative atoms in the times structure with the label goal. Furthermore, with each bottom-up application of the rule $\mathsf{b}{\downarrow}$, we extend the label of the produced action with a natural number that was not previously used. This way, we make sure that each bottom-up application of the rule $\mathsf{b}{\downarrow}$ to an action structure results in atoms with distinct labels, also in the case when the rule $\mathsf{b}{\downarrow}$ is applied to the same action structure more than once. We then read the constraints, containing the desired information, as follows: Whenever the rule $\mathsf{ai}{\downarrow}$ is applied bottom-up to a par structure, this results in a constraint that states that the label of the positive atom is ordered less than the label of the negative atom with respect to an ordering relation. Putting all these constraints together, we get a partial order. Now, let me express the above ideas formally.

DEFINITION 8.72. *Let $< \subseteq \mathscr{A} \times \mathscr{A}$ be a binary relation on a set $\mathscr{A}$. $<$ is a* strict partial order *if and only if it is irreflexive and transitive(, which implies asymmetry). A partially ordered set is also called a* poset*. The transitive reflexive reduction of a (strict) partial order is called the* cover relation*. An element z of a poset* covers *another element of x provided that there is no y in the poset for which $x < y < z$. In this case, z is called an* upper cover *of x and x a* lower cover *of z.*

DEFINITION 8.73. *Let* Π *be the proof*

$$\rho\,\frac{\overset{\Pi'}{\overset{\|}{S\{T\}}}\mathsf{ELS}}{S\{R\}}$$

of a cpps where the atoms in every action structure are labelled with the name of that action. Furthermore, whenever there is an instance of the rule $\mathsf{b}{\downarrow}$, the labels of the atoms in the premise, that are copied, are extended with a natural number that does not occur with the same action name elsewhere in the proof. Similarly, in a

problem structure, all the positive and negative atoms are labelled with init *(*i*) and* goal *(*g*), respectively. Let* Label *denote the set of all the labels occurring in* Π*. The function* μ *on* Π *is defined as follows.*

- *If* ρ *is an instance of the rule* $\mathsf{ai}{\downarrow}$ *where* R *is the structure* $[a_{\mathsf{l}}, \bar{a}_{\mathsf{k}}]$ *for an atom* a *such that* $\mathsf{l}, \mathsf{k} \in \mathsf{Label}$*, then*

$$\mu(\Pi) = \{\, (\mathsf{l}, \mathsf{k}) \,\} \cup \mu \begin{pmatrix} \Pi' \overset{\mathsf{ELS}}{\parallel} \\ S\{1\} \end{pmatrix} .$$

- *If* ρ *is an instance of a rule other than* $\mathsf{ai}{\downarrow}$ *and* $1{\downarrow}$*, then* $\mu(\Pi) = \mu(\Pi')$*.*
- *If* ρ *is the axiom* $1{\downarrow}$*, then* $\mu(\Pi) = \emptyset$*.*

Given a proof Π *of* $\mathcal{P}$*, a* constraint set *of* Π *for* $\mathcal{P}$*, denoted by* $\mathcal{C}_{\mathcal{P},\Pi}$*, is given with* $\mu(\Pi)$*. We will drop the subscripts when it is obvious from the context which cpps and proof we mean.*

Let us see the above definition on an example:

EXAMPLE 8.74. *Consider the conjunctive planning problem* $\mathscr{P}$ *of Example 8.5. The cpps* $\mathcal{P}$ *for* $\mathscr{P}$ *has a proof, which can be decomposed, by Corollary 8.25, as the following proof:*

$$\mathsf{ai}{\downarrow}\,\frac{\mathsf{ai}{\downarrow}\,\frac{\mathsf{ai}{\downarrow}\,\frac{\mathsf{ai}{\downarrow}\,\frac{\mathsf{ai}{\downarrow}\,\frac{1{\downarrow}\,\frac{}{1}}{!\,([d_{\mathsf{init}}, \bar{d}_{\mathsf{c_{euro}}}])}}{!\,([f_{\mathsf{c_{euro}}}, \bar{f}_{\mathsf{b_{candy}}}], [\bar{f}_{\mathsf{b_{candy}}}, f_{\mathsf{c_{euro}}}])}}{!\,([f_{\mathsf{c_{euro}}}, \bar{f}_{\mathsf{b_{lem}}}], [\bar{f}_{\mathsf{b_{candy}}}, f_{\mathsf{c_{euro}}}], [\bar{h}_{\mathsf{goal}}, h_{\mathsf{h_{lunch}}}])}}{!\,([f_{\mathsf{c_{euro}}}, \bar{f}_{\mathsf{b_{lem}}}], [\bar{f}_{\mathsf{b_{candy}}}, f_{\mathsf{c_{euro}}}], [\bar{h}_{\mathsf{goal}}, h_{\mathsf{h_{lunch}}}], [\bar{l}_{\mathsf{h_{lunch}}}, l_{\mathsf{b_{lem}}}])}}{!\,([f_{\mathsf{c_{euro}}}, \bar{f}_{\mathsf{b_{lem}}}], [\bar{f}_{\mathsf{b_{candy}}}, f_{\mathsf{c_{euro}}}], [\bar{h}_{\mathsf{goal}}, h_{\mathsf{h_{lunch}}}], [\bar{l}_{\mathsf{h_{lunch}}}, l_{\mathsf{b_{lem}}}], [\bar{c}_{\mathsf{h_{lunch}}}, c_{\mathsf{b_{can}}}])}}{!\,([d_{\mathsf{init}}, \bar{d}_{\mathsf{c_{euro}}}], [\bar{f}_{\mathsf{b_{lem}}}, f_{\mathsf{c_{euro}}}], [\bar{f}_{\mathsf{b_{candy}}}, f_{\mathsf{c_{euro}}}], [\bar{h}_{\mathsf{goal}}, h_{\mathsf{h_{lunch}}}], [\bar{l}_{\mathsf{h_{lunch}}}, l_{\mathsf{b_{lem}}}], [\bar{c}_{\mathsf{h_{lunch}}}, c_{\mathsf{b_{can}}}])}$$

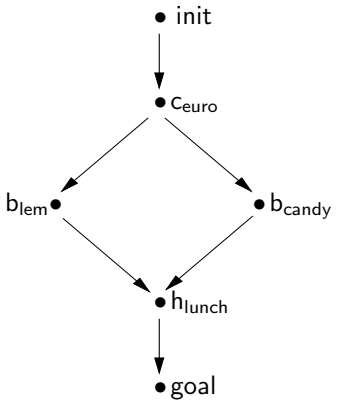
$$\Big\| \{\mathsf{s},\mathsf{p}{\downarrow},\mathsf{b}{\downarrow},\mathsf{w}{\downarrow}\}$$

$$\mathcal{P}$$

After plugging this proof into the function μ*, we get the following constraint set:*

$$\{(\mathsf{init}, \mathsf{c_{euro}}),\ (\mathsf{c_{euro}}, \mathsf{b_{lem}}),\ (\mathsf{c_{euro}}, \mathsf{b_{candy}}),\ (\mathsf{b_{lem}}, \mathsf{h_{lunch}}),\ (\mathsf{b_{candy}}, \mathsf{h_{lunch}}),\ (\mathsf{h_{lunch}}, \mathsf{goal})\} .$$

Observe that this constraint set gives the cover relation of a partial order of actions. This relation can be depicted as the following diagram, which overlaps with the diagram in the middle of Figure 8.7 when the nodes init *and* goal *are disregarded:*



REMARK 8.75. *It is important to note that although I used a decomposed proof of a cpps to extract a plan in the above example, this is not necessary to extract a constraint set. Thus, any proof of a cpps can be plugged into the function μ. Different proofs with the same instances of the rule* $\mathsf{ai}{\downarrow}$ *will deliver the same constraint set.*

A constraint set $\mathcal{C}$ is not necessarily a cover relation.

EXAMPLE 8.76. *Consider the planning problem given with*

$$\mathscr{A} = \{\quad \mathsf{a}_1 : \dot{\{}\, a \,\dot{\}} \to \dot{\{}\, c \,\dot{\}}\,, \qquad\qquad \mathsf{a}_2 : \dot{\{}\, b, c \,\dot{\}} \to \dot{\{}\, d \,\dot{\}}\ \ \}\,,$$

initial state $\mathcal{I} = \dot{\{}\, a, b \,\dot{\}}$ *and the goal world state* $\mathcal{G} = \dot{\{}\, d \,\dot{\}}$. *The cpps for this planning problem results in the constraint set*

$$\mathcal{C} = \{\, (\mathsf{init}, \mathsf{a}_1),\, (\mathsf{a}_1, \mathsf{a}_2),\, (\mathsf{a}_2, \mathsf{goal}),\, (\mathsf{init}, \mathsf{a}_2)\,\}$$

which is not a cover relation. The cover relation of $\mathcal{C}$ *is the set* $\mathcal{C}' \subset \mathcal{C}$ *given by*

$$\mathcal{C}' = \{\, (\mathsf{init}, \mathsf{a}_1),\, (\mathsf{a}_1, \mathsf{a}_2),\, (\mathsf{a}_2, \mathsf{goal})\,\}\,.$$

Let me now state some properties of constraint sets:

PROPOSITION 8.77. *Let* $\mathcal{C}_{\mathcal{P},\Pi}$ *be a constraint set of a proof* Π *for a cpps* $\mathcal{P}$.

(*i*) *There is no label* $\mathsf{x} \in \mathsf{Label}$, *such that* $(\mathsf{goal}, \mathsf{x}) \in \mathcal{C}$.
(*ii*) *There is no label* $\mathsf{x} \in \mathsf{Label}$, *such that* $(\mathsf{x}, \mathsf{init}) \in \mathcal{C}$.

PROOF. The result follows from the observation that (*i*) all the atoms that are labelled with goal are negative atoms, and (*ii*) all the atoms that are labelled with init are positive atoms. □

PROPOSITION 8.78. *Let* $\mathcal{P}$ *be a cpps and* $\mathcal{C}_{\mathcal{P},\Pi}$ *be the constraint set of a proof* Π *of* $\mathcal{P}$.

(*i*) $\mathcal{C}_{\mathcal{P},\Pi}$ *is antisymmetric.*
(*ii*) $\mathcal{C}_{\mathcal{P},\Pi}$ *is irreflexive.*

PROOF. (*i*) Assume that $\mathcal{C}$ is not antisymmetric, that is, for some $\mathsf{p}, \mathsf{q} \in \mathscr{A}$, $(\mathsf{p}, \mathsf{q}) \in \mathcal{C}$ and $(\mathsf{q}, \mathsf{p}) \in \mathcal{C}$. From Corollary 8.25, for some structure R, we must have

that Π decomposes to the following proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1}\,\mathsf{1}\downarrow}{[b_{\mathsf{q}},\bar{b}_{\mathsf{p}}]}\,\mathsf{ai}\downarrow}{([a_{\mathsf{p}},\bar{a}_{\mathsf{q}}],[b_{\mathsf{q}},\bar{b}_{\mathsf{p}}])}\,\mathsf{ai}\downarrow}}{\vdots}\,\mathsf{ai}\downarrow}{([a_{\mathsf{p}},\bar{a}_{\mathsf{q}}],[b_{\mathsf{q}},\bar{b}_{\mathsf{p}}],R)}\,\mathsf{s}}{([a_{\mathsf{p}},(\bar{a}_{\mathsf{q}},[b_{\mathsf{q}},\bar{b}_{\mathsf{p}}])],R)}\,\mathsf{s}}{([[a_{\mathsf{p}},\bar{b}_{\mathsf{p}}],(\bar{a}_{\mathsf{q}},b_{\mathsf{q}})],R)}$$
$$\Delta \Big\| \mathsf{ELS}$$
$$\mathcal{P}$$

This can be the case if the structure $[a_{\mathsf{p}},\bar{b}_{\mathsf{p}}]$ is produced by a conjunctive action structure corresponding to an action p, which contradicts with the definition of the conjunctive action structures.
(ii) Assume that there is a pair $(\mathsf{p},\mathsf{p}) \in \mathcal{C}$. Because two atoms can have the same label only if they are produced by the same conjunctive action structure by an application of the $\mathsf{b}\downarrow$ rule, where $R = (\bar{c}_1,\ldots,\bar{c}_m)$ and $T = [e_1,\ldots,e_n]$, there must be an action structure $(\bar{a}_{\mathsf{p}},R,[a_{\mathsf{p}},T])$, such that

$$\dfrac{\dfrac{}{1}\,\mathsf{1}\downarrow}{[a_{\mathsf{p}},\bar{a}_{\mathsf{p}}]}\,\mathsf{ai}\downarrow$$
$$\Delta \Big\|$$
$$[(\bar{a}_{\mathsf{p}},R,[a_{\mathsf{p}},T]),\bar{R},\bar{T}]$$

which is the case when there is a derivation Δ' such that

$$[a_{\mathsf{p}},\bar{a}_{\mathsf{p}}]$$
$$\Delta' \Big\|$$
$$(a_{\mathsf{p}},\bar{a}_{\mathsf{p}})\ .$$

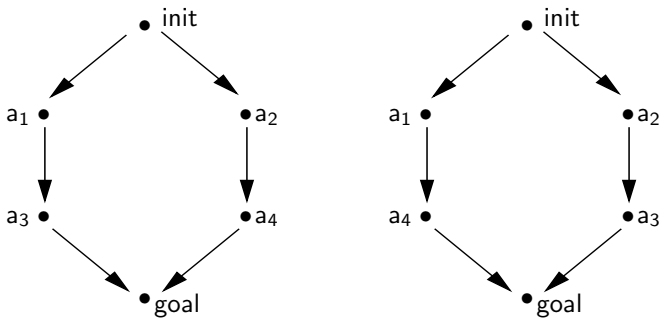
Because there cannot be such a derivation Δ', there cannot be a pair $(\mathsf{p},\mathsf{p}) \in \mathcal{C}$. □

REMARK 8.79. *Because a cpps $\mathcal{P}$ may have proofs that differ in the instances of the rule* $\mathsf{ai}\downarrow$*, it does not necessarily have a unique constraint set:*

EXAMPLE 8.80. *Consider the cpps $\mathcal{P}_1$ for the planning problem given with*

$$\mathscr{A} = \{\quad \mathsf{a}_1 : \dot{\{}\, a \,\dot{\}} \to \dot{\{}\, c \,\dot{\}}\,, \qquad \mathsf{a}_3 : \dot{\{}\, c \,\dot{\}} \to \dot{\{}\, d \,\dot{\}}\,,$$
$$\mathsf{a}_2 : \dot{\{}\, b \,\dot{\}} \to \dot{\{}\, c \,\dot{\}}\,, \qquad \mathsf{a}_4 : \dot{\{}\, c \,\dot{\}} \to \dot{\{}\, e \,\dot{\}}\ \ \}\,,$$

initial state $\mathcal{I} = \dot{\{}\, a,b \,\dot{\}}$ and the goal world state $\mathcal{G} = \dot{\{}\, d,e \,\dot{\}}$. Then the cpps $\mathcal{P}_1$ of this planning problem results in the two distinct constraint sets, which can be depicted as the two following diagrams:



However, as a consequence of Corollary 8.25, it is easy to observe that two different proofs of a cpps $\mathcal{P}$ have the same constraint set if they decompose to the same proof by permuting the rules, because they have the same instances of the $\mathsf{ai}{\downarrow}$ rule.

DEFINITION 8.81. *Let $\mathcal{P}$ be a cpps and $\mathcal{C}_{\mathcal{P},\Pi}$ be a constraint set of a proof Π for $\mathcal{P}$.*

(*i.*) *The* concurrent plan order of Π for $\mathcal{P}$, *denoted by* $\mathsf{Con}_{\mathcal{P},\Pi}$, *is the transitive reduction of* $\mathcal{C}_{\mathcal{P},\Pi}$.

(*ii.*) *The* securing order of Π for $\mathcal{P}$, *denoted by* $\mathsf{Sec}_{\mathcal{P},\Pi}$, *is the transitive closure of* $\mathcal{C}_{\mathcal{P},\Pi}$.

PROPOSITION 8.82. $\mathsf{Con}_{\mathcal{P},\Pi}$ *is a cover relation.*

PROOF. By Proposition 8.78, $\mathcal{C}_{\mathcal{P},\Pi}$ is an antisymmetric irreflexive relation, thus the transitive reduction of $\mathcal{C}_{\mathcal{P},\Pi}$ delivers a cover relation. □

PROPOSITION 8.83. $\mathsf{Sec}_{\mathcal{P},\Pi}$ *is a strict partial order.*

PROOF. By Proposition 8.78, $\mathcal{C}_{\mathcal{P},\Pi}$ is an antisymmetric irreflexive relation, thus the transitive closure of $\mathcal{C}_{\mathcal{P},\Pi}$ delivers a strict partial order. □

DEFINITION 8.84. *A* linearization Lin *of a securing order* Sec *is a strict total order defined on* Label, *such that* $\mathsf{Sec} \subseteq \mathsf{Lin}$. *Then a* plan P *induced by* Lin *is the sequence of actions that obeys the order defined by a linearization* Lin *of* Sec *so that, from left to right, the actions are sequenced from* init *to* goal, *excluding these two labels.*

EXAMPLE 8.85. *Returning to our running example, the concurrent plan order is* $\mathsf{Con} = \mathcal{C}$ *, given in Example 8.74, and the securing order* Sec *is the set*

$$\mathcal{C} \cup \{(\mathsf{init},\mathsf{goal}),\ (\mathsf{init},\mathsf{b}_{\mathsf{candy}}),\ (\mathsf{b}_{\mathsf{candy}},\mathsf{goal}),\ (\mathsf{init},\mathsf{h}_{\mathsf{lunch}}),$$
$$(\mathsf{c}_{\mathsf{euro}},\mathsf{goal}),\ (\mathsf{c}_{\mathsf{euro}},\mathsf{h}_{\mathsf{lunch}}),\ (\mathsf{init},\mathsf{b}_{\mathsf{lem}}),\ (\mathsf{b}_{\mathsf{lem}},\mathsf{goal})\}\,.$$

Then we get the two plans

$$\mathsf{P}_1 = \langle\, \mathsf{c}_{\mathsf{euro}};\ \mathsf{b}_{\mathsf{lem}};\ \mathsf{b}_{\mathsf{candy}};\ \mathsf{h}_{\mathsf{lunch}} \,\rangle \quad \textit{and} \quad \mathsf{P}_2 = \langle\, \mathsf{c}_{\mathsf{euro}};\ \mathsf{b}_{\mathsf{candy}};\ \mathsf{b}_{\mathsf{lem}};\ \mathsf{h}_{\mathsf{lunch}} \,\rangle\,.$$

LEMMA 8.86. *Let $\mathcal{C}$ be the constraint set of a proof Π for a cpps $\mathcal{P}$ and* Lin *be a linearization of the securing order* $\mathsf{Sec}_{\mathcal{P},\Pi}$. *For an action* $\mathsf{a} \in \mathsf{Label}$, *if* $(\mathsf{init},\mathsf{a}) \in \mathsf{Lin}$ *and* a *is the upper cover of* init *in* Lin, *then* $(\mathsf{init},\mathsf{a}) \in \mathcal{C}$ *and* a *is the upper cover of* init *in* $\mathcal{C}$.

PROOF. Observe that $\mathcal{C} \subseteq \mathsf{Sec} \subseteq \mathsf{Lin}$. Assume that $(\mathsf{init}, \mathsf{a}) \in \mathsf{Lin}$ and $(\mathsf{init}, \mathsf{a}) \notin \mathsf{Sec}$. This would imply that in Sec a and init are partially ordered, that is, there must be an action $\mathsf{a}' \in \mathsf{Label}$ such that $(\mathsf{a}', \mathsf{a}) \in \mathsf{Sec}$ and $(\mathsf{a}', \mathsf{init}) \in \mathsf{Sec}$. This contradicts with Proposition 8.77, so we have that $(\mathsf{init}, \mathsf{a}) \in \mathsf{Sec}$. Because a is the upper cover of init in Lin, it follows that a is an upper cover of init also in Sec. Because Sec is the transitive closure of $\mathcal{C}$, it follows that $(\mathsf{init}, \mathsf{a}) \in \mathcal{C}$ and a is the upper cover of init in $\mathcal{C}$, because otherwise a would not be the upper cover of init in Sec. □

THEOREM 8.87. *Let $\mathcal{P}$ be a cpps for a planning problem $\mathscr{P}$ such that there is a proof $\overset{\Pi}{\mathcal{P}}\!\!\prod^{\mathsf{ELS}}$. A plan P solves $\mathscr{P}$ if and only if plan P is induced by a linearization Lin of $\mathsf{Sec}_{\mathcal{P},\Pi}$.*

PROOF. Proof by induction on the length k of P.
($\Rightarrow$:) If $k = 0$ then it must be that $\mathcal{I} = \dot{\{}\, r_1, \ldots r_m \,\dot{\}} = \mathcal{G}$. Thus, the proof Π consists of an instance of the rule $\mathsf{termination}$ and $\mathcal{C}$ is $\{(\mathsf{init}, \mathsf{goal})\}$.

Turning to the induction step, for an action $\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$, let $\mathsf{P} = \langle\, \mathsf{a} \,;\, \mathsf{P}' \,\rangle$. With Corollary 8.28, we can assume the proof Π to be of the form

$$\mathsf{action}\ \frac{\overset{\Pi'}{\mathcal{P}}\!\!\prod^{\mathsf{ELS}}}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, ![r'_1, \ldots, r'_{m'}(\bar{g}_1, \ldots, \bar{g}_n)]]} \quad = \quad \mathsf{action}\ \frac{\overset{\Pi'}{\mathcal{P}'}\!\!\prod^{\mathsf{ELS}}}{\mathcal{P}}$$

where $\mathcal{P}'$ is the cpps for the planning problem that is solved by P'. It follows that

$$\mathcal{C}_{\mathcal{P}',\Pi'} = \big(\, \mathcal{C}_{\mathcal{P},\Pi} \cup \{(\mathsf{init}, x) \,|\, (\mathsf{a}, x) \in \mathcal{C}_{\mathcal{P},\Pi}\,\}\,\big) \setminus \big(\, \{(\mathsf{init}, \mathsf{a})\} \cup \{(\mathsf{a}, x) \,|\, (\mathsf{a}, x) \in \mathcal{C}_{\mathcal{P},\Pi}\,\}\,\big)\,.$$

Because $\mathsf{Sec}_{\mathcal{P},\Pi}$ and $\mathsf{Sec}'_{\mathcal{P}',\Pi'}$ are transitive closures of $\mathcal{C}_{\mathcal{P},\Pi}$ and $\mathcal{C}_{\mathcal{P}',\Pi'}$, respectively, we have that

$$\mathsf{Sec}_{\mathcal{P}',\Pi'} = \mathsf{Sec}_{\mathcal{P},\Pi} \setminus \big(\, \{(\mathsf{init}, \mathsf{a})\} \cup \{(\mathsf{a}, x) \,|\, (\mathsf{a}, x) \in \mathsf{Sec}_{\mathcal{P},\Pi}\,\}\,\big)\,.$$

From the induction hypothesis, we have that P' is induced by a linearization Lin' of $\mathsf{Sec}_{\mathcal{P}',\Pi'}$. Lin is obtained by adding pairs (x, y) to $\mathsf{Sec}_{\mathcal{P},\Pi}$ such that partially ordered nodes in $\mathsf{Sec}_{\mathcal{P},\Pi}$ become totally ordered in Lin. Thus, we can take

$$\mathsf{Lin} = \mathsf{Lin}' \cup \{(\mathsf{init}, \mathsf{a})\} \cup \{(\mathsf{a}, x) \,|\, (x, y) \in \mathsf{Lin}'\}$$

that induces $\langle\, \mathsf{a} \,;\, \mathsf{P}' \,\rangle$.
($\Leftarrow$:) If $k = 0$, then the constraint set $\mathcal{C}$ must be of the form $\{(\mathsf{init}, \mathsf{goal})\}$. This implies that $\mathcal{P}$ is of the form

$$[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, ![r_1, \ldots r_m, (\bar{r}_1, \ldots \bar{r}_m)]]$$

where $\mathcal{I} = \dot{\{}\, r_1, \ldots r_m \,\dot{\}} = \mathcal{G}$. Thus, the empty plan $\circ$ with length 0 solves $\mathscr{P}$.

Turning to the induction step, let $\mathsf{P} = \langle\, \mathsf{a} \,;\, \mathsf{P}' \,\rangle$ and Label denote the actions in P and Label' denote the actions in P'. That is, $\mathsf{Label} = \mathsf{Label}' \cup \{\mathsf{a}\}$ and $\mathsf{a} \notin \mathsf{Label}'$. If P is induced by Lin, it must be that

$$\mathsf{Lin} = \{(\mathsf{init}, \mathsf{a}), (\mathsf{a}, \mathsf{goal})\} \cup \{(\mathsf{a}, x) \,|\, x \in \mathsf{Label}'\} \cup \mathsf{Lin}'$$

where Lin' is a total order on $\mathsf{Label}' \cup \{\mathsf{init}, \mathsf{goal}\}$ such that P' is the plan induced by Lin'. This implies that there are two partitions $\mathsf{L}_1, \mathsf{L}_2$ of Label' such that $\mathsf{L}_1 \cup \mathsf{L}_2 =$

Label' and $\mathsf{L}_1 \cap \mathsf{L}_2 = \emptyset$ so that the following holds:

$$\mathsf{Sec}_{\mathcal{P},\Pi} = \{(\mathsf{init}, \mathsf{a}), (\mathsf{a}, \mathsf{goal})\} \cup \{(\mathsf{a}, x) \mid x \in \mathsf{L}_2\} \cup \mathsf{Sec}'$$

where Sec' is a strict partial order such that

$$\{(\mathsf{init}, x) \mid x \in \mathsf{L}_1\} \subseteq \mathsf{Sec}' \subseteq \mathsf{Lin}' .$$

With Lemma, 8.86 we have that $(\mathsf{init}, \mathsf{a}) \in \mathcal{C}$ and a is the upper cover of init in C. Thus, there must be instances of the rule $\mathsf{ai}{\downarrow}$ in Π from which $(\mathsf{init}, \mathsf{a})$ is extracted. This can only be the case when, for the action $\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \to \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$, there is an action structure A in $\mathcal{P}$ that interacts with the problem structure and for the initial state $\mathcal{I}$ we have that $\dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \dot{\subseteq} \mathcal{I}$. Assume that $\mathcal{I} = \dot{\{}\, c_1, \ldots, c_p, r_1, \ldots, r_m \,\dot{\}}$. It follows from Proposition 8.10 that we can construct a planning problem $\mathscr{P}'$ given with the same action set and goal state as $\mathscr{P}$ and the initial state $\mathcal{I}' = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$. Let $\mathcal{P}'$ be the cpps for $\mathscr{P}'$. Observe that the proof Π' of $\mathcal{P}'$ can be obtained from proof Π, by Corollary 8.28, as follows:

$$\mathsf{action}\ \frac{\Pi' \prod \mathsf{ELS}\ [?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,[r_1, \ldots, r_m(\bar{g}_1, \ldots, \bar{g}_n)]]}{[?\mathsf{A}_1, \ldots, ?\mathsf{A}_\mathsf{s}, !\,[c_1, \ldots, c_p, r_1, \ldots, r_m(\bar{g}_1, \ldots, \bar{g}_n)]]}$$

Because all the instance of the rule $\mathsf{ai}{\downarrow}$ in Π' are also instances of this rule in Π, it follows that $\mathsf{Sec}_{\mathcal{P},\Pi} \supset \mathsf{Sec}_{\mathcal{P}',\Pi'} = \mathsf{Sec}' \subseteq \mathsf{Lin}'$, and, with induction hypothesis, P' induced by Lin' solves $\mathscr{P}'$. Thus, $\mathsf{P} = \langle\, \mathsf{a}\,;\mathsf{P}' \,\rangle$ solves $\mathscr{P}$. □

COROLLARY 8.88. *Given a securing order* $\mathsf{Sec}_{\mathcal{P},\Pi}$ *of* Π *for the cpps* $\mathcal{P}$*, if* P *is a plan induced by a linearization* Lin *of* $\mathsf{Sec}_{\mathcal{P},\Pi}$*, then there is a securing* S *in* $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ *such that* $\mathsf{P} = \ell(\mathsf{S})$*.*

PROOF. It follows from Theorem 8.87 that P solves $\mathscr{P}$. From Corollary 8.28, there is a proof where the rule action for those actions that appear in the plan are applied in the same order and there is a successful path in $\mathsf{TS}[\![\mathcal{P}]\!]$ where exactly these actions are applied in order. It follows from Theorem 8.71 that this successful path provides a securing S in $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ such that $\mathsf{P} = \ell(\mathsf{S})$. □

COROLLARY 8.89. *Let* S *be a securing in* $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ *such that* $\mathcal{P} \overset{\ell(\mathsf{S})}{\twoheadrightarrow} \Delta$ *in* $\mathsf{TS}[\![\mathcal{P}]\!]$ *is a successful path. Then there is a proof* Π *of* $\mathcal{P}$ *with the instances of the rule* $\mathsf{b}{\downarrow}$ *as in* Δ *and there is a linearization* Lin *of* $\mathsf{Sec}_{\mathcal{P},\Pi}$ *that induces* $\ell(\mathsf{S})$*.*

PROOF. From Corollary 8.28 we have that every successful path in $\mathsf{TS}[\![\mathcal{P}]\!]$ corresponds to a plan P that solves the planning problem. It follows from Theorem 8.87 that there is a linearization Lin of $\mathsf{Sec}_{\mathcal{P},\Pi}$ that induces $\ell(\mathsf{S}) = \mathsf{P}$. □

REMARK 8.90. *So far, the notion of a securing order for a cpps* $\mathcal{P}$ *is defined on proofs that correspond to successful paths in* $\mathsf{TS}[\![\mathcal{P}]\!]$*. However, it is possible to generalize the notion of securing order to other derivations that correspond to arbitrary paths in* $\mathsf{TS}[\![\mathcal{P}]\!]$*. This can be done by modifying the premise of a derivation* Δ*, that is, by replacing the problem structure in the premise of* Δ *with a pseudo problem structure on which the rule* $\mathsf{termination}$ *can be applied. Applying the function* μ *of Definition 8.73 to this modified derivation delivers a securing order that is analogous to the securing order for successful paths.*

8.4. The Language $\mathcal{K}$

In this section, I will present the language $\mathcal{K}$. The language $\mathcal{K}$ is obtained by encoding the conjunctive planning problems in system NEL, similar to the encoding of the conjunctive planning problems in system ELS. However, in language $\mathcal{K}$, the parallel and sequential composition of the plans, respectively, are mapped to the commutative par operator and the non-commutative seq operator of this system. Thus, in a purely logical framework, without resorting to function symbols, this language brings sequential and parallel composition of the plans to the same level as in process algebras. This way, the structure of the planning problems and plans which solve these problems is captured by the logical connectives. Because the causality is expressed by means of resources as in the linear logic approach, the plans computed in the language $\mathcal{K}$ respect the LES semantics, also at the level of syntax. The structure of the plans captured by the logical operators makes it possible to perform logical reasoning on these plans.

8.4.1. The Syntax. In this subsection, I will present the syntax of the language $\mathcal{K}$ by means of an encoding of the conjunctive planning problems in system NEL.

Definition 8.91. *Given an action* $\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \rightarrow \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}$, *the* sequential action structure for a, *denoted by* Q *(possibly indexed), is a structure of the form*

$$\langle (\bar{c}_1, \ldots, \bar{c}_p); \mathsf{a}; [e_1, \ldots, e_q] \rangle \,.$$

An encoding of the conjunctive planning problems in NEL is as follows:

Definition 8.92. *Given a conjunctive planning problem* $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$, *let* $\mathsf{Q}_1, \ldots, \mathsf{Q}_s$ *be the sequential action structures for all the actions* $\mathsf{a}_1, \ldots, \mathsf{a}_s \in \mathscr{A}$ *and* K *be the problem structure for* $\mathcal{I}$ *and* $\mathcal{G}$. The sequential conjunctive planning problem structure (scpps) for $\mathscr{P}$, *denoted by* $\mathcal{R}$, *is defined as follows:*

$$[?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{K}]$$

Analogous to the encoding of the conjunctive planning planning problems in multiplicative exponential linear logic in Section 8.2, because an action can be executed arbitrarily many times, I employ the exponential "?". This retains a controlled contraction and weakening on the action structures. Thus, an action structure can be duplicated by applying the rule $\mathsf{b}{\downarrow}$ or annihilated by applying the rule $\mathsf{w}{\downarrow}$ during the search for the plans.

Example 8.93. *The scpps for the planning problem of Example 8.5 is as follows:*

$$[?\,\langle\, \bar{e}\,;\mathsf{c}_{\mathsf{euro}};[f,f]\rangle, ?\,\langle\, \bar{f}\,;\mathsf{b}_{\mathsf{lem}};l\rangle, ?\,\langle\, \bar{f}\,;\mathsf{b}_{\mathsf{candy}};c\rangle, ?\,\langle\, (\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle,\, e, \bar{h}\,] \,.$$

The structures $\langle\, \bar{e}\,;\mathsf{c}_{\mathsf{euro}};[f,f]\rangle$, $\langle\, \bar{f}\,;\mathsf{b}_{\mathsf{lem}};l\rangle$, $\langle\, \bar{f}\,;\mathsf{b}_{\mathsf{candy}};c\rangle$ *and* $\langle\, (\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle$, *respectively, are the sequential action structures for the actions* $\mathsf{c}_{\mathsf{euro}}$, $\mathsf{b}_{\mathsf{lem}}$, $\mathsf{b}_{\mathsf{can}}$, $\mathsf{h}_{\mathsf{lunch}}$, *respectively. The atom* e *denotes the initial state, and the atom* $\bar{h}$ *denotes the goal state.*

The non-commutative operator *seq* allows to capture the sequential composition of actions, thus it suffices to express plans which consist of sequentially composed actions. In the following, I will show that searching for certain kind of derivation of a scpps for a planning problem is equivalent to searching for a solution for this

planning problem. In such derivations, a plan solving the planning problem is delivered at the premise of the resulting derivation representing the computation. In these plans, it is possible to observe the parallel composition of the actions, which is mapped to the commutative *par* operator, at the same syntactic level as the sequential composition. Before presenting the operational semantics of the language $\mathcal{K}$ by means of the inference rules of system NEL, I would like to conclude the discussion on the syntax of this language. For this purpose, let me now formally define the plans where sequential and parallel composition of actions co-exist.

DEFINITION 8.94. *A* concurrent plan structure *is a structure generated by*

$$\mathsf{P^c} ::= \circ \mid \mathsf{a} \mid \langle\, \mathsf{P^c}\,;\, \mathsf{P^c} \,\rangle \mid [\, \mathsf{P^c}\,,\, \mathsf{P^c}\,]$$

where a *denotes atoms representing actions.*

8.4.2. Operational Semantics. Analogous to the rules action and $\mathsf{termination}$ of Section 8.2, the inference rules in the below definitions give the operational semantics of the language $\mathcal{K}$ for plans consisting of sequences of actions:

DEFINITION 8.95. *The following rule is called* sequential action*:*

$$\mathsf{action_{seq}}\ \frac{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle,\,\langle\mathsf{P};\mathsf{a};[E,R]\rangle]}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[c_1,\ldots,c_p,R]\rangle]}$$

LEMMA 8.96. *The rule* $\mathsf{action_{seq}}$ *is derivable for system* NEL.

PROOF. Take the following derivation where the instance of the rule $\mathsf{i}\!\downarrow$ is as given in Proposition 4.46:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};\mathsf{a};\,[E,R]\rangle]}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[\langle\mathsf{a};\,E\rangle,R]\rangle]}\,\mathsf{q}\!\downarrow}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[\langle\,[c_1,\ldots,c_p,(\bar{c}_1,\ldots,\bar{c}_p)];\mathsf{a};\,E\rangle,R]\rangle]}\,\mathsf{i}\!\downarrow}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[c_1,\ldots,c_p,\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle,R]\rangle]}\,\mathsf{q}\!\downarrow}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[c_1,\ldots,c_p,R]\rangle]}\,\mathsf{q}\!\downarrow}{S[\,?\langle\,(\bar{c}_1,\ldots,\bar{c}_p);\mathsf{a};\,E\rangle\,,\langle\mathsf{P};[c_1,\ldots,c_p,R]\rangle]}\,\mathsf{b}\!\downarrow$$

□

DEFINITION 8.97. *The following rule is called* sequential termination*:*

$$\mathsf{termination_{seq}}\ \frac{\mathsf{P}}{[?\mathsf{Q}_1,\ldots,?\mathsf{Q}_s,\langle\mathsf{P};[g_1,\ldots,g_m,(\bar{g}_1,\ldots,\bar{g}_m)]\rangle]}$$

LEMMA 8.98. *The rule* $\mathsf{termination_{seq}}$ *is derivable for system* NEL.

PROOF. Take the following derivation where the instance of the rule $\mathsf{i}\!\downarrow$ is as given in Proposition 4.46:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{P}}{[?\mathsf{Q}_s,\mathsf{P}]}\,\mathsf{w}\!\downarrow}{\vdots}\,\mathsf{w}\!\downarrow}{[?\mathsf{Q}_1,\ldots,?\mathsf{Q}_s,\mathsf{P}]}\,\mathsf{w}\!\downarrow}{[?\mathsf{Q}_1,\ldots,?\mathsf{Q}_s,\langle\mathsf{P};[g_1,\ldots,g_m,(\bar{g}_1,\ldots,\bar{g}_m)]\rangle]}\,\mathsf{i}\!\downarrow$$

□

By applying the rules action and termination bottom-up to scpps, it is possible to search for plans.

EXAMPLE 8.99. *Consider the scpps for the planning problem of Example 8.5. Let $S\{\ \}$ denote the structure context*

$$[?\langle \bar{e}\,;\mathsf{c_{euro}};[f,f]\rangle, ?\langle \bar{f}\,;\mathsf{b_{lem}};l\rangle, ?\langle \bar{f}\,;\mathsf{b_{candy}};c\rangle, ?\langle (\bar{l},\bar{c})\,;\mathsf{h_{lunch}};h\rangle, \{\ \}] \ .$$

The below derivation corresponds to a successful search for a plan:

$$\mathsf{termination}_{\mathsf{seq}}\frac{\langle \mathsf{c_{euro}};\mathsf{b_{lem}};\mathsf{b_{candy}};\mathsf{h_{lunch}}\rangle}{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};\mathsf{b_{candy}};\mathsf{h_{lunch}};[h,\bar{h}]\rangle}$$
$$\mathsf{action}_{\mathsf{seq}}\frac{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};\mathsf{b_{candy}};\mathsf{h_{lunch}};[h,\bar{h}]\rangle}{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};\mathsf{b_{candy}};[l,c,\bar{h}]\rangle}$$
$$\mathsf{action}_{\mathsf{seq}}\frac{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};\mathsf{b_{candy}};[l,c,\bar{h}]\rangle}{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};[l,f,\bar{h}]\rangle}$$
$$\mathsf{action}_{\mathsf{seq}}\frac{S\langle \mathsf{c_{euro}};\mathsf{b_{lem}};[l,f,\bar{h}]\rangle}{S\langle \mathsf{c_{euro}};[f,f,\bar{h}]\rangle}$$
$$\mathsf{action}_{\mathsf{seq}}\frac{S\langle \mathsf{c_{euro}};[f,f,\bar{h}]\rangle}{S[e,\bar{h}]}$$

The plan at the premise of this derivation is a solution for the corresponding planning problem.

In the following, I will prove a theorem that justifies the correctness of the above encoding with respect to (sequential) plans solving a conjunctive planning problem. Before this, let me collect some results which will be useful in the proof of this theorem. Let me first state a result which Straßburger proved in [**Str03a**].

THEOREM 8.100. (decomposition) *For every proof* $\prod_{R}$ *in system* NEL*, there are derivations* Δ_1,Δ_2*, and* Δ_3*, such that there is a derivation*

$$\begin{array}{c} \Delta_3 \Big\| \{\mathsf{ai}\downarrow,\mathsf{s},\mathsf{q}\downarrow,\mathsf{p}\downarrow\} \\ R_2 \\ \Delta_2 \Big\| \{\mathsf{w}\downarrow\} \\ R_1 \\ \Delta_1 \Big\| \{\mathsf{b}\downarrow\} \\ R \end{array}$$

for some structures R_1 and R_2.

PROPOSITION 8.101. *For every derivation* $\begin{array}{c} T \\ \Delta \,\|\, \mathsf{NEL} \\ R \end{array}$ *, there is a proof* $\begin{array}{c} \prod \mathsf{NEL} \\ [R,\bar{T}] \end{array}$ *.*

PROOF. Take the proof

$$\mathsf{i}\downarrow \frac{\circ\downarrow\dfrac{}{\circ}}{[T,\bar{T}]} \quad . \qquad \begin{array}{c} \\ \Delta \Big\| \mathsf{NEL} \\ [R,\bar{T}] \end{array}$$

□

COROLLARY 8.102. *Let* $\mathcal{R} = [?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{K}]$ *be a scpps and* P *be a plan that solves* $\mathcal{R}$. *For every derivation* $\dfrac{\mathsf{P}}{\mathcal{R}}\,\|\!|_{\mathsf{NEL}}$ *there are derivations* $\Delta_1, \Delta_2, \Delta_3$, *such that*

$$\begin{array}{c} \\ \Delta_3 \big\| \!\!\top_{\mathsf{BV}} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{K}, \bar{\mathsf{P}}] \\ \Delta_2 \,[\!|_{\{\mathsf{w}\downarrow\}} \\ [?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{K}, \bar{\mathsf{P}}] \\ \Delta_1 \,[\!|_{\{\mathsf{b}\downarrow\}} \\ [?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{K}, \bar{\mathsf{P}}] \end{array}$$

where for all $\mathsf{Q} \in \dot{\{}\, \mathsf{Q}_1, \ldots, \mathsf{Q}_k \,\dot{\}}$, *it holds that* $\mathsf{Q} \in \{\mathsf{Q}_1, \ldots, \mathsf{Q}_s\}$, *and there are* k *number of atoms in* P *that denote actions.*

PROOF. Follows immediately from Theorem 8.100 and Proposition 8.101. □

PROPOSITION 8.103. *Let* $R = [S\{\bar{a}\}, a]$ *be a* BV *structure that consists of pairwise distinct atoms.* R *has a proof in system* BV *if and only if* $S\{\circ\}$ *has a proof.*

PROOF. ($\Rightarrow$:) Construct a proof of $S\{\circ\}$ from the proof of R by replacing a and $\bar{a}$ with $\circ$. ($\Leftarrow$:) The proof follows from the derivation

$$\begin{array}{c} \mathsf{ai}{\downarrow}\dfrac{S\{\circ\}}{S[\bar{a}, a]} \\ \big[\!\big|_{\{\mathsf{s}\}} \\ [S\{\bar{a}\}, a] \end{array} .$$

□

PROPOSITION 8.104. *Let* $R = [\langle \bar{a}; P\rangle, \langle a; Q\rangle, U]$ *be a* BV *structure that consists of pairwise distinct atoms.* R *has a proof in system* BV *if and only if* $[P, Q, U]$ *has a proof.*

PROOF. ($\Rightarrow$:) Construct a proof of $[P, Q, U]$ from the proof of R by replacing a and $\bar{a}$ with $\circ$. ($\Leftarrow$:) The proof follows from the derivation

$$\begin{array}{c} \mathsf{ai}{\downarrow}\dfrac{[P, Q, U]}{[\langle [a, \bar{a}]; [P, Q]\rangle, U]} \\ \mathsf{q}{\downarrow}\dfrac{}{[\langle \bar{a}; P\rangle, \langle a; Q\rangle, U]} \end{array} .$$

□

THEOREM 8.105. *Let* $\mathscr{P} = \langle \mathcal{I}, \mathcal{G}, \mathscr{A}, \mathcal{F}\rangle$ *be a conjunctive planning problem and* $\mathcal{R}$ *be the scpps for* $\mathscr{P}$. *A plan* P *solves* $\mathscr{P}$ *if and only if there is a derivation* $\Delta \dfrac{\mathsf{P}}{\mathcal{R}}\,\|\!|_{\mathsf{NEL}}$.

PROOF. Proof by induction on the length k of plan P.
($\Rightarrow$:) Analogous to the proof of Theorem 8.27: for the base case apply Lemma 8.98, and for the inductive case apply Lemma 8.96.

($\Leftarrow$:) For the base case, if $k = 0$, then from Corollary 8.102, there must be a proof of the following form:

$$\dfrac{}{[r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]}\,\mathsf{BV}$$
$$\dfrac{[r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]}{[?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s,\ r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)]}\,[\![\{\mathsf{w}\downarrow, \mathsf{b}\downarrow\}$$

In order for such a proof to exist, it must be that $\dot{\{}\, r_1, \ldots, r_m \,\dot{\}} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$. Thus, the empty plan solves the planning problem $\mathscr{P}$.

Turning to the induction step we assume that the result holds for all plans with length k. Suppose that there is a planning problem $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$ where

$$\mathcal{I} = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}\,, \qquad \mathcal{G} = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}\ ,$$

$\mathcal{R} = [?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{K}]$ is the scpps for $\mathscr{P}$, and there is a derivation $\dfrac{\mathsf{P}}{\mathcal{R}}\,[\![\mathsf{NEL}$ where P is a plan. From Corollary 8.102, it follows that there must be a proof of the following form:

$$\dfrac{\Pi}{[\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{K}, \bar{\mathsf{P}}]}\,\mathsf{BV}$$
$$\dfrac{[\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{K}, \bar{\mathsf{P}}]}{[?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{Q}_1, \ldots, \mathsf{Q}_k,\ \mathsf{K}, \bar{\mathsf{P}}]}\,\Delta_2[\![\{\mathsf{w}\downarrow\}$$
$$\dfrac{[?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s, \mathsf{Q}_1, \ldots, \mathsf{Q}_k,\ \mathsf{K}, \bar{\mathsf{P}}]}{[?\mathsf{Q}_1, \ldots, ?\mathsf{Q}_s,\ \mathsf{K},\ \bar{\mathsf{P}}]}\,\Delta_1[\![\{\mathsf{b}\downarrow\}$$

Let Π' be the following proof obtained from the proof Π above by renaming the atoms in a way such that there are only structures that consist of pairwise distinct atoms at the premise and conclusion of each instance of the inference rules (and there are $k+1$ number of atoms in P denoting actions).

$$\dfrac{\Pi'}{[\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{Q}_{k+1}, r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n), \bar{\mathsf{P}}]}\,\mathsf{BV}$$

Then there must be an action $\mathsf{a} \in \mathscr{A}$ such that, for a plan P', $\mathsf{P} = \langle \mathsf{a}; \mathsf{P}' \rangle$ and

$$\mathsf{a} : \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \to \dot{\{}\, e_1, \ldots, e_q \,\dot{\}}\,. \tag{3}$$

Further, for every $r \in \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$, there must be action structures

$$\mathsf{Q}_i = \langle (\bar{c}_{i,1}, \ldots, \bar{c}_{i,p_i}); \mathsf{a}_i; [e_{i,1}, \ldots, e_{i,q_i}] \rangle \in \dot{\{}\, \mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{Q}_{k+1} \,\dot{\}}$$

such that $r \in \dot{\{}\, c_{i,1}, \ldots, c_{i,p_i} \,\dot{\}}$. Without loss of generality assume that $r \notin \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$. Because there cannot be a provable BV structure of the form

$$[\langle (\bar{c}, \ldots, \bar{c}); \mathsf{a}; [e, \ldots, e] \rangle, \ldots, \langle (\bar{c}, \ldots, \bar{c}); \mathsf{a}; [e, \ldots, e] \rangle, (\bar{g}, \ldots, \bar{g}), \bar{\mathsf{P}}]$$

it follows from Proposition 8.103 that there must be an action structure

$$\mathsf{Q} \in \dot{\{}\, \mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{Q}_{k+1} \,\dot{\}} \text{ such that}$$
$$\mathsf{Q} = \langle (\bar{c}_1, \ldots, \bar{c}_p); \mathsf{a}; [e_1, \ldots, e_q] \rangle \quad \text{and} \quad \dot{\{}\, c_1, \ldots, c_p \,\dot{\}} \mathrel{\dot{\subseteq}} \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}\,.$$

For $p \leq m$, let $\dot{\{}\, r_1, \ldots, r_p \,\dot{\}} = \dot{\{}\, c_1, \ldots, c_p \,\dot{\}}$ such that

$$\dot{\{}\, r_1, \ldots, r_m \,\dot{\}} = \dot{\{}\, c_1, \ldots, c_p, r_{p+1}, \ldots, r_m \,\dot{\}} \text{ and}$$

$$\dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}} = \dot{\{}\, r_{p+1}, \ldots, r_m \,\dot{\}} \,\dot{\cup}\, \dot{\{}\, e_1, \ldots, e_q \,\dot{\}} \text{ where } m' = m - p + q.$$

Because of commutativity and associativity we can assume that $\mathsf{Q} = \mathsf{Q}_{k+1}$. By applying Proposition 8.103 we get the following proof:

$$\begin{array}{c} \Pi'' \Big\| \mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \langle \mathsf{a}; [e_1, \ldots, e_q] \rangle, r_{p+1}, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n), \langle \bar{\mathsf{a}}; \bar{\mathsf{P}}' \rangle] \\ [\!| \mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \langle (c_1, \ldots, c_p); \mathsf{a}; [e_1, \ldots, e_q] \rangle, r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n), \langle \bar{\mathsf{a}}; \bar{\mathsf{P}}' \rangle] \end{array}$$

By applying Proposition 8.104 to proof Π'' above we obtain the following proof:

$$\begin{array}{c} \Big\| \mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k, e_1, \ldots, e_q, r_{p+1}, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n), \bar{\mathsf{P}}'] \end{array}$$

It follows from the induction hypothesis that P' solves the planning problem $\mathscr{P}' = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}', \mathcal{G} \rangle$ where

$$\mathcal{I}' = \dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}} = \dot{\{}\, e_1, \ldots, e_q, r_{p+1}, \ldots, r_m \,\dot{\}}\,.$$

Thus, from (3), it follows that $\langle \mathsf{a}; \mathsf{P}' \rangle$ solves $\mathscr{P}$. □

Remark 8.106. *Similar to the encoding of conjunctive planning problems in* ELS*, presented in Section 8.2, the encoding in* NEL *requires the state reached at the end of the computation to be strictly equal to the goal state. However, similar to the ideas stated in Remark 8.30, this condition can be relaxed by introducing action structures that consume the excessive resources without producing any new resource. For instance, for each resource $r \in \mathscr{R}$, one can define an action that has only this resource as the condition and an empty effect. In the encoding, such an action is represented by a negated atom $\bar{r}$ for each $r \in \mathscr{R}$.*

Example 8.107. *Consider the planning problem of Example 8.31. We get the following scpps for this planning problem*

$$[?\,\langle \bar{e}; \mathsf{c}_{\mathsf{euro}}; [f, f] \rangle, ?\,\langle \bar{f}; \mathsf{b}_{\mathsf{lem}}; l \rangle, ?\,\bar{f},\ e, \bar{l}\,]$$

which results in the following derivation:

$$\cfrac{\cfrac{\cfrac{\cfrac{\langle \mathsf{c}_{\mathsf{euro}}; \mathsf{b}_{\mathsf{lem}} \rangle}{[?\langle \bar{e}; \mathsf{c}_{\mathsf{euro}}; [f, f] \rangle, ?\langle \bar{f}; \mathsf{b}_{\mathsf{lem}}; l \rangle, ?\bar{f},\ \langle \mathsf{c}_{\mathsf{euro}}; \mathsf{b}_{\mathsf{lem}}; [l, \bar{l}] \rangle\,]}\,\mathsf{termination}_{\mathsf{seq}}}{[?\langle \bar{e}; \mathsf{c}_{\mathsf{euro}}; [f, f] \rangle, ?\langle \bar{f}; \mathsf{b}_{\mathsf{lem}}; l \rangle, ?\bar{f},\ \langle \mathsf{c}_{\mathsf{euro}}; \mathsf{b}_{\mathsf{lem}}; [l, f, \bar{l}] \rangle\,]}\,\mathsf{action}_{\mathsf{seq}}}{[?\langle \bar{e}; \mathsf{c}_{\mathsf{euro}}; [f, f] \rangle, ?\langle \bar{f}; \mathsf{b}_{\mathsf{lem}}; l \rangle, ?\bar{f},\ \langle \mathsf{c}_{\mathsf{euro}}; [f, f, \bar{l}] \rangle\,]}\,\mathsf{action}_{\mathsf{seq}}}{[?\langle \bar{e}; \mathsf{c}_{\mathsf{euro}}; [f, f] \rangle, ?\langle \bar{f}; \mathsf{b}_{\mathsf{lem}}; l \rangle, ?\bar{f},\ e, \bar{l}\,]}\,\mathsf{action}_{\mathsf{seq}}$$

8.4.3. Concurrent Plans. So far we have seen that for plans consisting of sequences of actions the operational semantics of language $\mathcal{K}$ can be given by means of the inference rules of system NEL. In the following, we will see that the inference rules of system NEL provide also the operational semantics of language $\mathcal{K}$ for concurrent plans: By means of the inference rules of system NEL, we can compute a concurrent plan as the premise of a derivation representing the computation.

DEFINITION 8.108. *The following rule is called* sequential composition*:*

$$\mathsf{sequential}\ \frac{S\langle C;\mathsf{P}_1;\mathsf{P}_2;[E_1,E_2]\rangle}{S[\langle C;\mathsf{P}_1;[r_1,\ldots,r_m,E_1]\rangle,\langle(\bar{r}_1,\ldots,\bar{r}_m);\mathsf{P}_2;E_2\rangle]}$$

LEMMA 8.109. *The rule* sequential *is derivable for system* BV*.*

PROOF. Take the following derivation where the instance of the rule i↓ is as given in Proposition 4.46:

$$\dfrac{\dfrac{\dfrac{\dfrac{S\langle C;\mathsf{P}_1;\mathsf{P}_2;[E_1,E_2]\rangle}{S\langle C;\mathsf{P}_1;[\langle\mathsf{P}_2;E_2\rangle,E_1]\rangle}\,\mathsf{q}\downarrow}{S\langle C;\mathsf{P}_1;[\langle[r_1,\ldots,r_m,(\bar{r}_1,\ldots,\bar{r}_m)];\mathsf{P}_2;E_2\rangle,E_1]\rangle}\,\mathsf{i}\downarrow}{S\langle C;\mathsf{P}_1;[r_1,\ldots,r_m,\langle(\bar{r}_1,\ldots,\bar{r}_m);\mathsf{P}_2;E_2\rangle,E_1]\rangle}\,\mathsf{q}\downarrow}{S[\langle C;\mathsf{P}_1;[r_1,\ldots,r_m,E_1]\rangle,\langle(\bar{r}_1,\ldots,\bar{r}_m);\mathsf{P}_2;E_2\rangle]}\,\mathsf{q}\downarrow$$

□

DEFINITION 8.110. *The following rule is called* parallel composition*:*

$$\mathsf{parallel}\ \frac{S\langle(C_1,C_2);[\mathsf{P}_1,\mathsf{P}_2];[E_1,E_2]\rangle}{S[\langle C_1;\mathsf{P}_1;E_1\rangle,\langle C_2;\mathsf{P}_2;E_2\rangle]}$$

LEMMA 8.111. *The rule* parallel *is derivable for system* BV*.*

PROOF. Take the following derivation:

$$\dfrac{\dfrac{\dfrac{S\langle(C_1,C_2);[\mathsf{P}_1,\mathsf{P}_2];[E_1,E_2]\rangle}{S\langle[C_1,C_2];[\mathsf{P}_1,\mathsf{P}_2];[E_1,E_2]\rangle}\,\mathsf{s}}{S\langle[C_1,C_2];[\langle\mathsf{P}_1;E_1\rangle,\langle\mathsf{P}_2;E_2\rangle]\rangle}\,\mathsf{q}\downarrow}{S[\langle C_1;\mathsf{P}_1;E_1\rangle,\langle C_2;\mathsf{P}_2;E_2\rangle]}\,\mathsf{q}\downarrow$$

□

DEFINITION 8.112. *A concurrent plan* P^{c} solves a planning problem $\mathscr{P}$ *if, for all the derivations* $\begin{array}{l}\mathsf{P}\\ \|_{\{\mathsf{q}\downarrow\}}\\ \mathsf{P}^{\mathsf{c}}\end{array}$ *where* P *is a plan,* P *solves* $\mathscr{P}$*.*

EXAMPLE 8.113. *Consider the following derivation with the sccps* $\mathcal{R}$ *of the planning problem of Example 8.5:*

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle\mathsf{c}_{\mathsf{euro}}\,;[\mathsf{b}_{\mathsf{lem}}\,,\mathsf{b}_{\mathsf{candy}}];\mathsf{h}_{\mathsf{lunch}}\rangle}{\|_{\mathsf{BV}}}}{[e,\langle\,\bar{e}\,;\mathsf{c}_{\mathsf{euro}};[\mathsf{b}_{\mathsf{lem}},\mathsf{b}_{\mathsf{candy}}];\mathsf{h}_{\mathsf{lunch}};h\rangle,\bar{h}]}\,\mathsf{sequential}}{[e,\langle\,\bar{e}\,;\mathsf{c}_{\mathsf{euro}};[\mathsf{b}_{\mathsf{lem}},\mathsf{b}_{\mathsf{candy}}];[l,c]\rangle,\langle\,(\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle,\bar{h}]}\,\mathsf{sequential}}{[e,\langle\,\bar{e}\,;\mathsf{c}_{\mathsf{euro}};[f,f]\rangle,\langle\,(\bar{f},\bar{f})\rangle\,;[\mathsf{b}_{\mathsf{lem}},\mathsf{b}_{\mathsf{candy}}];[l,c]\,,\langle\,(\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle,\bar{h}]}\,\mathsf{parallel}$$

$$\dfrac{[e,\langle\,\bar{e}\,;\mathsf{c}_{\mathsf{euro}};[f,f]\rangle,\langle\,\bar{f}\,;\mathsf{b}_{\mathsf{lem}};l\rangle,\langle\,\bar{f}\,;\mathsf{b}_{\mathsf{candy}};c\rangle,\langle\,(\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle,\bar{h}]}{\|_{\{\mathsf{w}\downarrow,\mathsf{b}\downarrow\}}}$$

$$[?\,\langle\,\bar{e}\,;\mathsf{c}_{\mathsf{euro}};[f,f]\rangle,?\,\langle\,\bar{f}\,;\mathsf{b}_{\mathsf{lem}};l\rangle,?\,\langle\,\bar{f}\,;\mathsf{b}_{\mathsf{candy}};c\rangle,?\,\langle\,(\bar{l},\bar{c})\,;\mathsf{h}_{\mathsf{lunch}};h\rangle,\,e,\bar{h}\,]$$

The premise of this derivation is the concurrent plan structure

$$\langle\mathsf{c}_{\mathsf{euro}}\,;[\mathsf{b}_{\mathsf{lem}}\,,\mathsf{b}_{\mathsf{candy}}];\mathsf{h}_{\mathsf{lunch}}\rangle$$

which solves this planning problem.

In the following, I will show that searching for a concurrent plan structure that solves a conjunctive planning problem is equivalent to searching for a derivation as in the above Example. Let me first state a lemma that will be useful for showing this formally.

LEMMA 8.114. *Let* $\mathcal{I}_1 = \dot{\{}\, r_1, \ldots, r_m \,\dot{\}}$, $\mathcal{I}_2 = \dot{\{}\, r'_1, \ldots, r'_{m'} \,\dot{\}}$, $\mathcal{Z}_1 = \dot{\{}\, g_1, \ldots, g_n \,\dot{\}}$, $\mathcal{Z}_2 = \dot{\{}\, g'_1, \ldots, g'_{n'} \,\dot{\}}$, *and* $\mathcal{Z} = \mathcal{Z}_1 \dot{\cup} \mathcal{Z}_2$ *be states, and* $\mathsf{P}_1 = \langle \mathsf{a}_1; \ldots; \mathsf{a}_k \rangle$, $\mathsf{P}_2 = \langle \mathsf{a'}_1; \ldots; \mathsf{a'}_{k'} \rangle$ *be plans. Furthermore, let* $\mathsf{Q}_1, \ldots, \mathsf{Q}_k, \mathsf{Q'}_1, \ldots, \mathsf{Q'}_{k'}$ *be the sequential action structures for the actions* $\mathsf{a}_1, \ldots, \mathsf{a}_k, \mathsf{a'}_1, \ldots, \mathsf{a'}_{k'}$.

(*i*) $\Phi(\mathsf{P}_1, \mathcal{I}_1) = \mathcal{Z}_1$ *and* $\Phi(\mathsf{P}_2, \mathcal{I}_2) = \mathcal{Z}_2$.

(*ii*) $\Phi(\,\mathsf{P}_1, \Phi(\,\mathsf{P}_2, \mathcal{I}_1 \dot{\cup} \mathcal{I}_2\,)) = \Phi(\,\mathsf{P}_2, \Phi(\,\mathsf{P}_1, \mathcal{I}_1 \dot{\cup} \mathcal{I}_2\,)) = \mathcal{Z}$.

(*iii*) *There are the following derivations:*

$$\begin{array}{c} \mathsf{P}_1 \\ [\!|\mathsf{BV} \\ [r_1, \ldots, r_m, \mathsf{Q}_1, \ldots, \mathsf{Q}_k, (\bar{g}_1, \ldots, \bar{g}_n)] \end{array} \qquad \begin{array}{c} \mathsf{P}_2 \\ [\!|\mathsf{BV} \\ [r'_1, \ldots, r'_{m'}, \mathsf{Q'}_1, \ldots, \mathsf{Q'}_{k'}, (\bar{g'}_1, \ldots, \bar{g'}_{n'})] \end{array}$$

PROOF.

(i) $\Rightarrow$ (ii) : Follows immediately from Proposition 8.11.

(ii) $\Rightarrow$ (iii) : Follows immediately from Theorem 8.105.

(iii) $\Rightarrow$ (i) : The following derivations together with Theorem 8.105 prove the result.

$$\mathsf{q}{\downarrow}\,\frac{\langle \mathsf{P}_1; \mathsf{P}_2 \rangle}{[\mathsf{P}_1, \mathsf{P}_2]} \qquad\qquad \mathsf{q}{\downarrow}\,\frac{\langle \mathsf{P}_2; \mathsf{P}_1 \rangle}{[\mathsf{P}_1, \mathsf{P}_2]}$$

□

THEOREM 8.115. *Let* $\mathscr{P}$ *be a planning problem and* $\mathcal{R}$ *be the scpps for a planning problem* $\mathscr{P}$. $\mathsf{P^c}$ *is a concurrent plan that solves* $\mathscr{P}$ *if and only if there is a derivation of the following form:*

$$\begin{array}{c} \mathsf{P^c} \\ [\!|\mathsf{NEL} \\ \mathcal{R} \end{array}$$

PROOF. ($\Rightarrow$:) Let $\mathsf{P} = \langle \mathsf{a_1}; \ldots; \mathsf{a_k} \rangle$ be a plan such that there is a derivation $\begin{array}{c} \mathsf{P} \\ [\!|\{\mathsf{q}{\downarrow}\}} \\ \mathsf{P^c} \end{array}$. Let $\mathsf{Q}_1, \ldots, \mathsf{Q}_k$ be the sequential action structures for the actions $\mathsf{a_1}, \ldots, \mathsf{a_k}$. From Theorem 8.105 there is a derivation $\begin{array}{c} \langle \mathsf{a_1}; \ldots; \mathsf{a_k} \rangle \\ [\!|\mathsf{NEL} \\ \mathcal{R} \end{array}$. From Corollary 8.102, it follows that there is a derivation of the form

$$\begin{array}{c} \langle \mathsf{a_1}; \ldots; \mathsf{a_k} \rangle \\ {\scriptstyle \Delta_1}\,[\!|\mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k, r_1, \ldots, r_m, (\bar{g}_1, \ldots, \bar{g}_n)] \\ {\scriptstyle \Delta_2}\,[\!|\{\mathsf{w}{\downarrow}, \mathsf{b}{\downarrow}\}} \\ \mathcal{R} \end{array} .$$

From Δ_1, let us construct a derivation Δ such that we have a derivation of the following form:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{P}^c}{\mathsf{i}\downarrow\dfrac{}{\langle \mathsf{P}^c; [g_1, \ldots, g_n, (\bar{g}_1, \ldots, \bar{g}_n)]\rangle}}}{\mathsf{i}\downarrow\dfrac{}{\langle [r_1, \ldots, r_m, (\bar{r}_1, \ldots, \bar{r}_m)]; \mathsf{P}^c; [g_1, \ldots, g_n, (\bar{g}_1, \ldots, \bar{g}_n)]\rangle}}}{\mathsf{q}\downarrow\dfrac{}{[\langle [r_1, \ldots, r_m, (\bar{r}_1, \ldots, \bar{r}_m)]; \mathsf{P}^c; [g_1, \ldots, g_n]\rangle, (\bar{g}_1, \ldots, \bar{g}_n)]}}}{\mathsf{q}\downarrow\dfrac{}{[r_1, \ldots, r_m, \langle (\bar{r}_1, \ldots, \bar{r}_m); \mathsf{P}^c; [g_1, \ldots, g_n]\rangle, (\bar{g}_1, \ldots, \bar{g}_n)]}}}{\begin{array}{c}\Delta \Big\| \mathsf{BV} \\ [r_1, \ldots, r_m, \mathsf{Q}_1, \ldots, \mathsf{Q}_k, (\bar{g}_1, \ldots, \bar{g}_n)]\end{array}}$$

We will construct the derivation Δ with structural induction on P^c. Base case where $\mathsf{P}^c = \circ$ or $\mathsf{P}^c = \mathsf{a}$ being trivial let us consider the inductive cases:

– If $\mathsf{P}^c = [\mathsf{P}^c_1, \mathsf{P}^c_2]$ then there must be two concurrent plans $\langle \mathsf{P}^c_1; \mathsf{P}^c_2\rangle$ and $\langle \mathsf{P}^c_2; \mathsf{P}^c_1\rangle$ that solve $\mathscr{P}$. This implies that there are sequential plans P_1 and P_2 with $\begin{array}{c}\mathsf{P}_1 \\ [\![\{\mathsf{q}\downarrow\}\end{array}$ $\begin{array}{c}\mathsf{P}^c_1\end{array}$ and $\begin{array}{c}\mathsf{P}_2 \\ [\![\{\mathsf{q}\downarrow\}\end{array}$ $\begin{array}{c}\mathsf{P}^c_2\end{array}$ such that $\mathsf{P} = \langle \mathsf{P}_1; \mathsf{P}_2\rangle$ or $\mathsf{P} = \langle \mathsf{P}_2; \mathsf{P}_1\rangle$. From Lemma 8.114 and Corollary 8.102, it follows that there exists planning problems with the following derivations:

$$\begin{array}{c}\mathsf{P}_1 \\ [\![\mathsf{BV} \\ [r_1, \ldots, r_{m'}, \mathsf{Q}_1, \ldots, \mathsf{Q}_{k'}, (\bar{g}_1, \ldots, \bar{g}_{n'})]\end{array} \qquad \begin{array}{c}\mathsf{P}_2 \\ [\![\mathsf{BV} \\ [r_{m'+1}, \ldots, r_m, \mathsf{Q}_{k'+1}, \ldots, \mathsf{Q}_k, (\bar{g}_{n'+1}, \ldots, \bar{g}_n)]\end{array}$$

Then, with the induction hypothesis, we get the derivations

$$\begin{array}{c}\langle (\bar{r}_1, \ldots, \bar{r}_{m'}); \mathsf{P}^c_1; [g_1, \ldots, g_{n'}]\rangle \\ \Delta_1 [\![\mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_{k'}]\end{array} \quad \text{and} \quad \begin{array}{c}\langle (\bar{r}_{m'+1}, \ldots, \bar{r}_m); \mathsf{P}^c_2; [g_{n'+1}, \ldots, g_n]\rangle \\ \Delta_2 [\![\mathsf{BV} \\ [\mathsf{Q}_{k'+1}, \ldots, \mathsf{Q}_k]\end{array} .$$

With Lemma 8.111, we can then construct the derivation Δ as follows:

$$\mathsf{parallel}\ \dfrac{\langle (\bar{r}_1, \ldots, \bar{r}_m); [\mathsf{P}^c_1, \mathsf{P}^c_2]; [g_1, \ldots, g_n]\rangle}{[\langle (\bar{r}_1, \ldots, \bar{r}_{m'}); \mathsf{P}^c_1; [g_1, \ldots, g_{n'}]\rangle, \langle (\bar{r}_{m'+1}, \ldots, \bar{r}_m); \mathsf{P}^c_2; [g_{n'+1}, \ldots, g_n]\rangle]}$$
$$\begin{array}{c}[\Delta_1, \Delta_2] \Big\| \mathsf{BV} \\ [\mathsf{Q}_1, \ldots, \mathsf{Q}_k]\end{array}$$

– If $\mathsf{P}^c = \langle \mathsf{P}^c_1; \mathsf{P}^c_2\rangle$ then there must be sequential plans P_1 and P_2 with $\begin{array}{c}\mathsf{P}_1 \\ [\![\{\mathsf{q}\downarrow\}\end{array}$ $\begin{array}{c}\mathsf{P}^c_1\end{array}$ and $\begin{array}{c}\mathsf{P}_2 \\ [\![\{\mathsf{q}\downarrow\}\end{array}$ $\begin{array}{c}\mathsf{P}^c_2\end{array}$ such that $\mathsf{P} = \langle \mathsf{P}_1; \mathsf{P}_2\rangle$.
From Proposition 8.10 and Corollary 8.102, it follows that, for some $f_1, \ldots, f_s \in \mathscr{R}$, there exists planning problems with the following derivations:

$$\begin{array}{c}\mathsf{P}_1 \\ [\![\mathsf{BV} \\ [r_1, \ldots, r_m, \mathsf{Q}_1, \ldots, \mathsf{Q}_{k'}, (\bar{f}_1, \ldots, \bar{f}_s)]\end{array} \qquad \begin{array}{c}\mathsf{P}_2 \\ [\![\mathsf{BV} \\ [f_1, \ldots, f_s, \mathsf{Q}_{k'+1}, \ldots, \mathsf{Q}_k, (\bar{g}_1, \ldots, \bar{g}_n)]\end{array}$$

Then, with the induction hypothesis, we get the derivations

$$\begin{array}{c}\langle(\bar{r}_1,\ldots,\bar{r}_m);\mathsf{P}^{\mathsf{c}}_1;[f_1,\ldots,f_s]\rangle\\ \mathbin{[\![}\mathsf{BV}\\ [\mathsf{Q}_1,\ldots,\mathsf{Q}_{k'}]\end{array} \quad \text{and} \quad \begin{array}{c}\langle(\bar{f}_1,\ldots,\bar{f}_s);\mathsf{P}^{\mathsf{c}}_2;[g_1,\ldots,g_n]\rangle\\ \mathbin{[\![}\mathsf{BV}\\ [\mathsf{Q}_{k'+1},\ldots,\mathsf{Q}_k]\end{array} \quad .$$

With Lemma 8.109, we can then construct the derivation Δ as follows:

$$\mathsf{sequential}\,\frac{\langle(\bar{r}_1,\ldots,\bar{r}_m);\langle\mathsf{P}^{\mathsf{c}}_1;\mathsf{P}^{\mathsf{c}}_2\rangle;[g_1,\ldots,g_n]\rangle}{[\langle(\bar{r}_1,\ldots,\bar{r}_{m'});\mathsf{P}^{\mathsf{c}}_1;[f_1,\ldots,f_s]\rangle,\langle(\bar{f}_1,\ldots,\bar{f}_s);\mathsf{P}^{\mathsf{c}}_2;[g_1,\ldots,g_n]\rangle]}$$
$$[\Delta_1,\Delta_2]\,\Big\|\,\mathsf{BV}$$
$$[\mathsf{Q}_1,\ldots,\mathsf{Q}_k]$$

($\Leftarrow$:) Follows immediately from Definition 8.112 and Theorem 8.105. □

8.4.4. Labeled Event Structure Semantics of Language $\mathcal{K}$. The similarities between the systems ELS and NEL and the encoding of the conjunctive planning problems in these systems allow to carry the results of Section 8.3 to the language $\mathcal{K}$. In particular, we can observe the LES semantics of the plans computed in this language by carrying the LES semantics of the cpps to scpps.

Remark 8.116. *We can associate a LES to the scpps $\mathcal{R}$ of a conjunctive planning problem $\mathscr{P}$ by applying the procedure for the cpps, described in Section 8.3, analogously to scpps: by replacing the rule* action *in Definition 8.33 with the rule* $\mathsf{action}_{\mathsf{seq}}$ *of Definition 8.95, we obtain a transition system* $\mathsf{TS}[\![\mathcal{R}]\!]$ *for the scpps $\mathcal{R}$. Similarly, by replacing the rule* termination *in Definition 8.36 and Definition 8.45 with the rule* $\mathsf{termination}_{\mathsf{seq}}$ *of Definition 8.97, we carry the discussions of Section 8.3 to scpps. This way, we obtain a labelled event structure* $\mathsf{LES}^\star[\![\mathcal{R}]\!]$ *for a scpps $\mathcal{R}$ that is isomorphic to the* $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ *for the cpps $\mathcal{P}$ of $\mathscr{P}$. Thus, from now on, for a conjunctive planning problem $\mathscr{P}$ with the cpps $\mathcal{P}$ and scpps $\mathcal{R}$, I will use the expressions* $\mathsf{LES}^\star[\![\mathcal{P}]\!]$ *and* $\mathsf{LES}^\star[\![\mathcal{R}]\!]$ *synonymously.*

Definition 8.117. *Let $\mathcal{R}$ be a scpps and P^{c} be a concurrent plan such that there is a derivation*

$$\begin{array}{c}\mathsf{P}^{\mathsf{c}}\\ \Delta\,\big\|\,\mathsf{NEL}\\ \mathcal{R}\end{array} \quad .$$

Let Π be the proof obtained from Δ by replacing each atom a *representing an action, in Δ, with the unit $\circ$. The* constraint set of Δ for $\mathcal{R}$*, denoted by $\mathcal{C}_{\mathcal{R},\Delta}$, is given by $\mu(\Pi)$.*

Definition 8.118. *Let $\mathcal{R}$ be a scpps and $\mathcal{C}_{\mathcal{R},\Delta}$ be a constraint set of a derivation Δ for $\mathcal{R}$.*

(*i.*) *The* concurrent plan order of Δ for $\mathcal{P}$*, denoted by $\mathsf{Con}_{\mathcal{R},\Delta}$, is the transitive reduction of $\mathcal{C}_{\mathcal{R},\Delta}$.*

(*ii.*) *The* securing order of Δ for $\mathcal{R}$*, denoted by $\mathsf{Sec}_{\mathcal{R},\Delta}$, is the transitive closure of $\mathcal{C}_{\mathcal{R},\Delta}$.*

Theorem 8.119. *Let $\mathcal{R}$ be a scpps for a planning problem $\mathscr{P}$ such that there is a derivation* $\begin{array}{c}\mathsf{P}^{\mathsf{c}}\\ \Delta\,\big\|\,\mathsf{NEL}\\ \mathcal{R}\end{array}$ *. There is a derivation* $\begin{array}{c}\mathsf{P}\\ \big\|\,\{\mathsf{q}\downarrow\}\\ \mathsf{P}^{\mathsf{c}}\end{array}$ *if and only if plan* P *is induced by a linearization* Lin *of* $\mathsf{Sec}_{\mathcal{R},\Delta}$*.*

PROOF. Analogous to the proof of Theorem 8.87. □

COROLLARY 8.120. *Given a securing order* $\mathsf{Sec}_{\mathcal{R},\Delta}$ *of* Δ *for the scpps* $\mathcal{R}$*, if* P *is a plan induced by a linearization* Lin *of* $\mathsf{Sec}_{\mathcal{R},\Delta}$*, then there is a securing* S *in* $\mathsf{LES}^{\star}[\![\mathcal{P}]\!]$ *such that* $\mathsf{P} = \ell(\mathsf{S})$.

PROOF. Analogous to the proof of Corollary 8.88. □

COROLLARY 8.121. *Let* S *be a securing in* $\mathsf{LES}^{\star}[\![\mathcal{R}]\!]$ *such that* $\mathcal{R} \overset{\ell(\mathsf{S})}{\twoheadrightarrow} \Delta$ *in* $\mathsf{TS}[\![\mathcal{R}]\!]$ *is a successful path. There is a derivation* $\Delta \big\|^{\mathsf{P}}_{\mathcal{R}} \mathsf{NEL}$ *and a linearization* Lin *of* $\mathsf{Sec}_{\mathcal{R},\Delta}$ *that induces* $\mathsf{P} = \ell(\mathsf{S})$.

PROOF. Analogous to the proof of Corollary 8.89. □

DEFINITION 8.122. *Given a securing order* Sec*, two plans* P_1 *and* P_2 *are* Sec-equivalent *if* P_1 *and* P_2*, respectively, are plans induced by linearizations* Lin_1 *and* Lin_2 *of* Sec.

COROLLARY 8.123. *Let* Sec *be a securing order. For any state* $\mathcal{Z}$*, if plans* P_1 *and* P_2 *are* Sec*-equivalent then* $\Phi(\mathsf{P}_1, \mathcal{Z}) = \Phi(\mathsf{P}_2, \mathcal{Z})$.

PROOF. Follows immediately from Theorem 8.87 and Theorem 8.119. □

8.4.5. Concurrent Computations in Language $\mathcal{K}$. So far, we have seen that there is a strict correspondence between the plans computed in language $\mathcal{K}$ and securings in the labelled event structures of the conjunctive planning problems. We have also seen that the securing orders (and the concurrent plan orders which are transitive reductions of securing orders) give canonical representations of sets of securings that correspond to plans solving planning problems. Further, we have seen that, analogously, a concurrent plan structure P^{c} gives a canonical representation of a set of plans P, determined by all the derivations $\Delta \big\|^{\mathsf{P}}_{\mathsf{P}^{\mathsf{c}}} \{\mathsf{q}\downarrow\}$.

DEFINITION 8.124. *A partial order* $\leq \subseteq \mathscr{A} \times \mathscr{A}$ *is* N-free (series-parallel) *if and only if, for all* $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \mathscr{A}$*,* $\{(\mathsf{a}, \mathsf{b}), (\mathsf{c}, \mathsf{d}), (\mathsf{c}, \mathsf{b})\} \subseteq \leq$ *implies* $(\mathsf{a}, \mathsf{d}) \in \leq$*. The* N-free closure *of a partial order* $\leq$ *is the smallest N-free partial order containing* $\leq$.

A concurrent plan structure provides a syntactical representation of a partial order of actions for alternative plans that solve a planning problem. In contrast to the securing orders, such a representation sets boundaries to the computations being modeled. These boundaries are meaningful from the point of view of concurrent computations: A partial order which is represented by a concurrent plan structure is an N-free partial order.

EXAMPLE 8.125. *Consider the partial orders denoted by the graphs below: The one on the left is an N-free partial order, whereas the one on the right is not.*

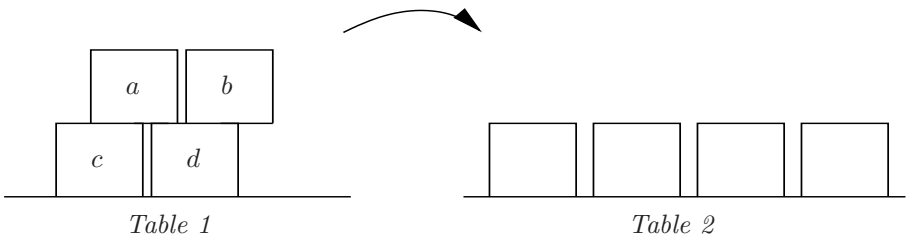


FIGURE 8.8. A planning problem

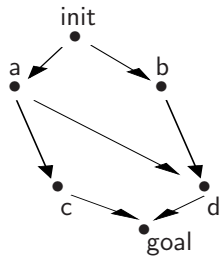
From the point of view of concurrent computations, a representation of computations as N-free partial orders is meaningful: When meet and join of two processes are considered as points in time, these N-free partial orders provide a representation of synchronization of processes. That is, because the representation of resources provides a model of dependencies, processes with a common meet and join can be executed concurrently. However, such an observation is impossible in a partial order that is not N-free. A securing order is not necessarily an N-free order. Thus, although a securing order provides a canonical representation of a class of plans which solve a planning problem, N-free closures of securing orders needs to be considered when modeling concurrent computations. Because the concurrent plan structures allow the representation of only N-free partial orders, they are well suited for modeling concurrent computations.

EXAMPLE 8.126. *Consider the following modification of the example planning problem of Chapter 1. As before, on Table 1 there are four blocks, which are stacked on top of each other, as shown on the left-hand side of the Figure 8.8. An action takes a block from Table 1 and puts it on Table 2. Because block a is stacked on blocks c and d, blocks c and d cannot be moved before block a. Similarly, block d cannot be moved before block b. However, this time the goal of the problem is moving all the four blocks from Table 1 to Table 2. We can express this scenario as the conjunctive planning problem* $\mathscr{P} = \langle \mathscr{R}, \mathscr{A}, \mathcal{I}, \mathcal{G} \rangle$ *where*

$$\mathcal{I} = \dot{\{}\, c_l, a_l, a_r, b_l, b_r \,\dot{\}}\,, \quad \mathcal{G} = \dot{\{}\, g, g, g, g \,\dot{\}} \text{ and}$$

$$\mathscr{A} = \{ \quad \mathsf{a} : \dot{\{}\, a_l, a_r \,\dot{\}} \to \dot{\{}\, c_r, d_l, g \,\dot{\}}\,, \qquad \mathsf{c} : \dot{\{}\, c_l, c_r \,\dot{\}} \to \dot{\{}\, g \,\dot{\}}\,,$$
$$\mathsf{b} : \dot{\{}\, b_l, b_r \,\dot{\}} \to \dot{\{}\, d_r, g \,\dot{\}}\,, \qquad \mathsf{d} : \dot{\{}\, d_l, d_r \,\dot{\}} \to \dot{\{}\, g \,\dot{\}}\ \}\ .$$

For a block x, the resource x_l *and* x_r*, respectively, denote that the left-hand side and the right-hand side, respectively, on top of the block x is free. Thus, in order for a block x to be moved, on top of this block both left-hand side and right-hand side must be free, i.e., both of the resources* x_l *and* x_r *must be available. The resource g denotes a block on Table 2. When we consider the scpps* $\mathcal{R}$ *of this problem, for any derivation* Δ *that delivers a solution for this problem, the constraint set* $\mathcal{C}_{\mathcal{R},\Delta}$ *of* Δ *for* $\mathcal{R}$ *is depicted as follows:*



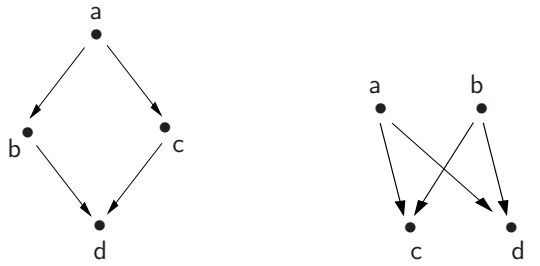
In this graph, we observe that the actions a *and* b *are partially ordered because they are independent from each other due to the resources that they require to be executed. Similarly the pairs* c, d *and* b, c *are partially ordered. Because such partially ordered actions can be executed in any order, this graph provides a canonical representation of the following plans:*

$$\langle \mathsf{a};\mathsf{b};\mathsf{c};\mathsf{d}\rangle \qquad \langle \mathsf{a};\mathsf{b};\mathsf{d};\mathsf{c}\rangle \qquad \langle \mathsf{a};\mathsf{c};\mathsf{b};\mathsf{d}\rangle \qquad \langle \mathsf{b};\mathsf{a};\mathsf{c};\mathsf{d}\rangle \qquad \langle \mathsf{b};\mathsf{a};\mathsf{d};\mathsf{c}\rangle$$

However, if one considers the concurrent executions, we observe that if the actions a *and* b *are executed concurrently, then* b *and* c *cannot be executed concurrently because* c *requires* a *to be executed. In this system, the possible concurrent executions are the ones that are given by the below concurrent plan structures. The sccps* $\mathcal{R}$ *provides derivations that result in these two concurrent plan structures in the premise.*

$$\langle \mathsf{a};[\mathsf{b},\mathsf{c}];\mathsf{d}\rangle \qquad\qquad \langle [\mathsf{a},\mathsf{b}];[\mathsf{c},\mathsf{d}]\rangle$$

It is important to observe that these concurrent plan structures denote N-free closures of the partial order that is given by the constraint set $\mathcal{C}_{\mathcal{R},\Delta}$ *above. (These N-free partial orders are that of relation* $\lhd_{\langle \mathsf{a};[\mathsf{b},\mathsf{c}];\mathsf{d}\rangle}$ *and* $\lhd_{\langle [\mathsf{a},\mathsf{b}];[\mathsf{d},\mathsf{c}]\rangle}$*.) Their graphical representations are depicted as the following Hasse-diagrams, respectively:*



Given that a concurrent plan structure gives a canonical representation of a set of plans solving a conjunctive planning problem, we can consider such a set of plans as an equivalence class of plans: A member of an equivalence class can be replaced with another one in any planning context, because all the members of such an equivalence class consumes and produces the same multiset of resources.

DEFINITION 8.127. *Given a concurrent plan structure* P^{c}*, plans* P_1 *and* P_2 *are* P^{c}-equivalent *if there are the derivations* $\mathsf{P}_1 \atop \Delta\!\!\left[\!\!\left[\{\mathsf{q}\downarrow\}\right.\right. \atop \mathsf{P}^{\mathsf{c}}$ *and* $\mathsf{P}_2 \atop \Delta\!\!\left[\!\!\left[\{\mathsf{q}\downarrow\}\right.\right. \atop \mathsf{P}^{\mathsf{c}}$.

LEMMA 8.128. *For* BV *structures which do not contain any copar structures, the rule* $\mathsf{ai}\downarrow$ *permutes over the rule* $\mathsf{q}\downarrow$.

PROOF. It suffices to check the cases excluded by the conditions of Remark 5.49. The case where the redex of $\mathsf{q}{\downarrow}$ is inside an active structure of the contractum of $\mathsf{ai}{\downarrow}$ is impossible because the contractum of the rule $\mathsf{ai}{\downarrow}$ is $\circ$. If the contractum of $\mathsf{ai}{\downarrow}$ is inside an active structure of the redex of $\mathsf{q}{\downarrow}$, then we permute as follows:

$$\mathsf{ai}{\downarrow}\dfrac{\mathsf{q}{\downarrow}\dfrac{S\langle[\langle R';R''\rangle,U];[T,V]\rangle}{S[\langle R';R'';T\rangle,\langle U;V\rangle]}}{S[\langle R';[a,\bar{a}];R'';T\rangle,\langle U;V\rangle]} \quad\rightsquigarrow\quad \mathsf{q}{\downarrow}\dfrac{\mathsf{ai}{\downarrow}\dfrac{S\langle[\langle R';R''\rangle,U];[T,V]\rangle}{S\langle[\langle R';[a,\bar{a}];R''\rangle,U];[T,V]\rangle}}{S[\langle R';[a,\bar{a}];R'';T\rangle,\langle U;V\rangle]}$$

□

REMARK 8.129. *As we have seen in Example 5.56, for* BV *structures in general, the rule* $\mathsf{ai}{\downarrow}$ *cannot be permuted over the other rules.*

THEOREM 8.130. *For a concurrent plan* P^{c} *and a plan* P *there is a derivation* $\Delta\dfrac{\mathsf{P}}{\mathsf{P}^{\mathsf{c}}}\{\mathsf{q}{\downarrow}\}$ *if and only if there is a proof* $\Pi\dfrac{}{[\mathsf{P}^{\mathsf{c}},\bar{\mathsf{P}}]}\{\circ{\downarrow},\mathsf{ai}{\downarrow},\mathsf{q}{\downarrow}\}$.

PROOF. ($\Rightarrow$:) Analogous to the proof of Proposition 8.101. ($\Leftarrow$:) Let $\mathsf{P} = \langle \mathsf{a}_1;\ldots;\mathsf{a}_n\rangle$. In proof Π, starting from the top-most instance of the rule $\mathsf{ai}{\downarrow}$, which appears below an instance of the rule $\mathsf{q}{\downarrow}$, permute all the instances of the rule $\mathsf{ai}{\downarrow}$ over the instances of the rule $\mathsf{q}{\downarrow}$ inductively to obtain a proof of the following form for some structure R:

$$\begin{array}{c}\dfrac{}{R}\{\mathsf{ai}{\downarrow}\}\\ \Delta'\dfrac{}{[\mathsf{P}^{\mathsf{c}},\bar{\mathsf{P}}]}\{\mathsf{q}{\downarrow}\}\end{array}$$

From Proposition 5.9, it R must be of the form $\langle[\mathsf{a}_1,\bar{\mathsf{a}}_1];\ldots;[\mathsf{a}_n,\bar{\mathsf{a}}_1]\rangle$, because for each a_i, it must be that $\mathsf{a}_i \downarrow \bar{\mathsf{a}}_i$, and a_i and $\bar{\mathsf{a}}_i$ must be in the same context in order for an instance of the rule $\mathsf{ai}{\downarrow}$ to annihilate them. Thus, the derivation obtained from the derivation Δ' by replacing $\bar{\mathsf{P}}$ with $\langle\circ;\ldots;\circ\rangle$ delivers the derivation Δ. □

COROLLARY 8.131. *For a concurrent plan structure* P^{c}*, plans* P_1 *and* P_2 *are* P^{c}-equivalent *if and only if there are proofs* $\dfrac{}{[\mathsf{P}^{\mathsf{c}},\bar{\mathsf{P}}_1]}\{\circ{\downarrow},\mathsf{ai}{\downarrow},\mathsf{q}{\downarrow}\}$ *and* $\dfrac{}{[\mathsf{P}^{\mathsf{c}},\bar{\mathsf{P}}_2]}\{\circ{\downarrow},\mathsf{ai}{\downarrow},\mathsf{q}{\downarrow}\}$.

PROOF. Follows immediately from Theorem 8.130. □

COROLLARY 8.132. *Let* P^{c} *be a concurrent plan structure. For any state* $\mathcal{Z}$*, if plans* P_1 *and* P_2 *are* P^{c}*-equivalent then* $\Phi(\mathsf{P}_1,\mathcal{Z}) = \Phi(\mathsf{P}_2,\mathcal{Z})$.

PROOF. Follows immediately from Theorem 8.115 and Theorem 8.130. □

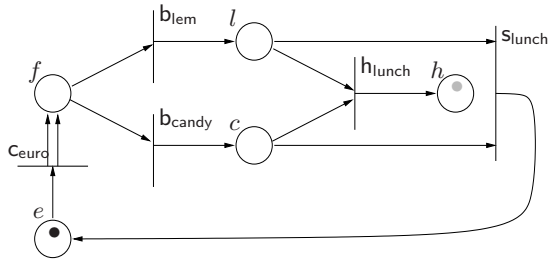
REMARK 8.133. *Let* P^{c} *be a concurrent plan structure. At an instance of the rule* $\mathsf{q}{\downarrow}$ *of the form* $\mathsf{q}{\downarrow}\dfrac{R}{\mathsf{P}^{\mathsf{c}}}$ *the structure* R *is a concurrent plan structure. It follows from Remark 5.13 and Proposition 5.14 that the length of a derivation* $\dfrac{\mathsf{P}}{\mathsf{P}^{\mathsf{c}}}\{\mathsf{q}{\downarrow}\}$ *is bounded by* $\mathcal{O}(|\mathsf{occ}\,\mathsf{P}^{\mathsf{c}}|^2)$.

8.5. Relation to Other Work

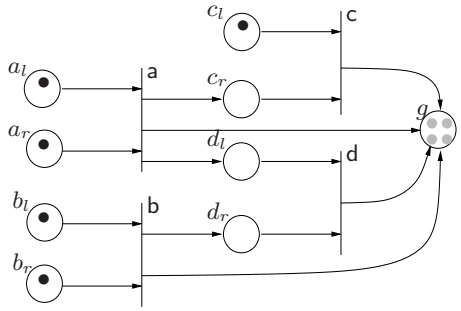
Reasoning about actions and planning, also from the point of view of conjunctive planning, has been studied by various authors. In this section, I will discuss the approach of this thesis in comparison to related work.

8.5.1. Expressive Power. In conjunctive planning, states are defined over the data structure multiset. Actions are considered as multiset rewriting rules. Also in the context of linear logic, it was previously shown that the multiset rewriting approach is complete for representing computations of certain classes of Petri nets [**Pet62**] (see, e.g., [**Asp87, GG89, MOM91, EW94, Cer95, IH01**]). In such an encoding, the multiset rewrite rules represent the possible firings of the transitions of a Petri net. The places of the net are represented by elements of multisets. Such a view allows to consider a conjunctive planning problem as the reachability problem of the corresponding Petri net and vice versa.

EXAMPLE 8.134. *The planning problem of Example 8.39 is depicted as the following Petri net. The token • represents the initial state and the token • represents the goal state.*



EXAMPLE 8.135. *Similarly, the planning problem of Example 8.126 is depicted as the following net:*



The reachability problem in Petri nets is known to be EXPSPACE-hard [**Lip76**]. Thus, the encoding of Petri nets in multiplicative exponential linear logic delivers the lower bound of this logic to be EXPSPACE-hard [**MOM91**]. When the complexity of a language is seen as a measure of expressive power, this also sets the scene for the expressive power of the propositional languages based on multiset rewriting in comparison to propositional languages based on STRIPS: Given that planning in STRIPS is PSPACE-complete [**Byl92**], because PSPACE is a strict subset of EXPSPACE the language $\mathcal{K}$ is strictly more expressive than propositional languages based on STRIPS. In order to achieve the same expressive power, the STRIPS language must be enriched with a constant-only first order language, i.e.,

DATALOG-STRIPS. The reason for this can be seen as follows: In the STRIPS language, the so called *pre-condition-lists*, *add-lists*, and *delete-lists* of an action are sets. However, in conjunctive planning, conditions and effects of an action are multisets. Multisets allow multiple occurrences of resources in the conditions and effects of the actions. In a propositional setting, such a representation cannot be achieved by sets over a finite set of constant symbols. For instance, consider the action $\mathsf{c}_{\mathsf{euro}}$ of Example 8.39:

$$\mathsf{c}_{\mathsf{euro}} \ : \ \dot{\{}\, e \,\dot{\}} \rightarrow \dot{\{}\, f, f \,\dot{\}}$$

Such an action cannot be represented in STRIPS unless we define a constant for one f, a constant for two f, another for three f, and so on.

However, a characterization of STRIPS in conjunctive planning is possible. In [**Kün03**], Küngas gives an encoding of the STRIPS planning problems within linear logic planning domains: A STRIPS action with the pre-condition-list PRE, delete-list DEL, and add-list ADD is translated into a multiset rewriting rule of the following form:

$$\mathsf{PRE} \ \rightarrow \ \mathsf{ADD} \cup (\, \mathsf{PRE} \setminus \mathsf{DEL} \,)$$

Because STRIPS lacks a clear logical semantics (see, e.g., [**Lif86**]), this translation assumes that for all the actions it holds that $\mathsf{DEL} \subseteq \mathsf{PRE}$.

If we consider the planning languages based on the situation calculus semantics, where worlds are described by means of properties, we see that any planning problem expressed in these languages can be expressed as a conjunctive planning problem: In [**Thi94**], Thielscher shows that conjunctive planning languages can be employed to encode the domain descriptions of the action description language $\mathcal{A}$ [**GL93**]. In this language, because the representation scheme is based on properties rather than resources, states are given by sets instead of multisets. Atomic properties of the world are represented by fluents. Because conjunctive planning domains do not support explicit negation, the translation of the domain descriptions is achieved by using two different resources for each fluent name, once representing the fluent affirmatively and once negatively. Consistency of the states and actions is guaranteed by disallowing the resources for a fluent to occur both negatively and affirmatively in a multiset. In these multisets, a resource is not allowed to occur more than once.

As it is stated in [**GL98**], because the action description language is equivalent to the propositional fragment of the planning language ADL [**Ped89**], the result of [**Thi94**] also implies that the language $\mathcal{K}$ can be used for ADL domains.

When planning problems are considered from the point of view of concurrent computations, due to the explicit representation of resources, the language $\mathcal{K}$ allows to observe *true concurrency* in the computations: In a language with true concurrency, when two actions are partially ordered, the outcome of their execution in parallel is same as the outcome of their execution in either order. As we have seen in Section 8.3, the explicit treatment of resources provides a representation of independence and causality. When two actions are partially ordered in a LES, in an execution that involves both of these actions, they are independent in terms of the resources that they require to be executed. Thus, their parallel composition results in an action that has the same effect as their execution in any order. The inference rule parallel of Definition 8.110 implements such a parallel composition.

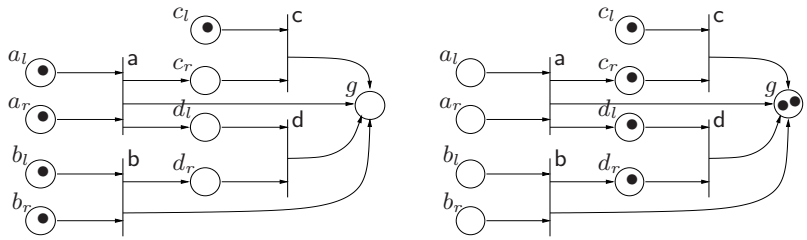
EXAMPLE 8.136. *Consider the conjunctive planning problem of Example 8.126. We have seen that the concurrent plan structure* $\langle[\mathsf{a},\mathsf{b}];[\mathsf{c},\mathsf{d}]\rangle$ *solves this planning problem. Let us consider the parallel composition of the actions* a *and* b *(or similarly, the actions* c *and* d*):*

$$\mathsf{parallel}\ \frac{\langle(a_l,a_r,b_l,b_r);[\mathsf{a},\mathsf{b}];[c_r,d_l,d_r,g,g]\rangle}{[\langle(a_l,a_r);\mathsf{a};[c_r,d_l,g]\rangle,\langle(b_l,b_r);\mathsf{b};[d_r,g]\rangle]}$$

The actions a *and* b *are independent, in the sense that output of one action is not the input of the other. Thus, the execution of the resulting action in the premise of the above derivation is equivalent to executing these two actions in either order.*

In the light of the above observations, it is easy to see that a concurrent plan structure provides a model of a true-concurrent computations of the corresponding Petri net.

EXAMPLE 8.137. *For the concurrent plan structure* $\langle[\mathsf{a},\mathsf{b}];[\mathsf{c},\mathsf{d}]\rangle$*, which solves the conjunctive planning problem of Example 8.126, the nets on the left-hand side and right-hand side below, respectively, demonstrate the state of the net before and after the execution of the concurrent action* $\alpha=[\mathsf{a},\mathsf{b}]$*, respectively:*



However, when planning problems are modeled by means of properties as in STRIPS or ADL, it is not always possible to observe true concurrency in the partial order plans computed by the planners for these languages, e.g., UCPOP [**PW92**], and Graphplan [**BM97**]. A simple modification of the famous dining philosophers problem is helpful to see the reason for this:

EXAMPLE 8.138. *There are two hungry philosophers,* a *and* b*, sitting at a dinner table. In order for a philosopher to eat, she must have a fork. However, there is only one fork on the table. The problem consists in finding a plan where both philosophers have eaten. The solution of this problem is a plan in which* a *and* b *eat in either order. A plan where* a *and* b *eat concurrently cannot be a solution for this problem, because* a *and* b *cannot have the fork at the same time. Because the fork is a resource, which cannot be shared, eating of one is dependent on the other's finishing eating and leaving the fork. Hence, these two actions can be executed in either order but not in parallel. A simple encoding of this scenario as a conjunctive planning problem allows to observe such a semantics:*

$$\mathcal{I}=\dot{\{}\,h_a,h_b,f\,\dot{\}}\,,\quad \mathcal{G}=\dot{\{}\,e_a,e_b,f\,\dot{\}}$$

$$\mathscr{A}=\{\ \mathsf{a}:\dot{\{}\,h_a,f\,\dot{\}}\to\dot{\{}\,e_a,f\,\dot{\}}\,,\qquad \mathsf{b}:\dot{\{}\,h_b,f\,\dot{\}}\to\dot{\{}\,e_b,f\,\dot{\}}\ \ \}$$

In the above encoding, for a philosopher x*, the resource* h_x *denotes that* x *is hungry and* e_x *denotes that* x *has eaten.* f *denotes the resource fork. The actions* a *and* b

can be executed in either order. However, their parallel composition results in the action

$$[\mathsf{a},\mathsf{b}] : \dot{\{}\, h_a, h_b, f, f \,\dot{\}} \to \dot{\{}\, e_a, e_b, f, f \,\dot{\}}$$

which requires two instances of the resource f in order to be executed. Thus the parallel composition of these two actions cannot be executed in the initial state $\mathcal{I}$. An encoding of this problem by means of properties, in a propositional language, in a way which delivers such a semantics is not straight-forward, if not impossible.

For an indepth exposure and related references on the relationship between the partial order characterization of process expressions, process algebras, and Petri nets, the reader is referred to [**BB00**], i.e., Chapter 13 of [**BPS01**].[4]

The relationship between conjunctive planning and Petri nets has been studied by various authors: In his PhD thesis [**Leh02**], Heiko Lehmann establishes a relationship between the conjunctive planning version of the fluent calculus [**HS90**] and the Petri nets in order to address decidability issues related to fluent calculus. There models of the fluent calculus domains are characterized as *labelled transition systems*. These transition systems are then used to introduce bisimulation on the models of fluent calculus domains. The computations that are modeled in [**Leh02**] are interleaving computations. In contrast, the LES semantics of the language $\mathcal{K}$, together with the concurrent plan structures, provides a model of non-interleaving (true-concurrent) computations.

By resorting to the correspondence between the Petri net reachability problem and the conjunctive planning problems, in [**Kün03**], Küngas presents an implementation for linear logic planning. He compares the performance of his planner, called RAPS, on STRIPS planning domains with several state-of-the-art domain-independent planners, provides experimental results, and gives references to related work. In [**Kün05**], Küngas uses the conjunctive planning to carry the abstraction techniques from planning to Petri nets. He shows that the upper computational complexity bound for Petri net reachability checking can be made polynomial by using abstraction hierarchies.

8.5.2. Other Approaches to Conjunctive Planning. Resource conscious deductive planning, based on multisets, has been elaborated both in the lines of fluent calculus and linear logic. [**BHS+93**] extends the conjunctive planning problems to handle disjunction of facts and this way express nondeterministic actions. There it is shown that this extended approach and the approach based on linear logic augmented by employing additives of linear logic in [**MTV90**] are equivalent with respect to the semantics of so called *disjunctive planning problems*.

In [**HT93**], Hölldobler and Thielscher study the specificity of the actions such that a more specific description of an action is preferred over a less specific description when it is applicable and whenever an applicable and most specific description is executed in a consistent situation then the resulting situation is also consistent. In [**EHT96**], the conjunctive planning approach was extended to cover hierarchical planning where actions are treated as resources that can be consumed at different levels of the planning process. This approach resorts to a chemical metaphor, adapted from concurrency theory, in which situations are represented as solutions in which floating molecules can interact freely according to interaction rules. During a reaction two such molecules are consumed and a new one is produced. A

[4] [**Bae04**] is an excellent survey on the history of process algebras.

planning problem is then formalized by a solution modeling the initial situation, a goal situation and the question of whether there exists a sequence of interactions (plan) transforming the initial situation into a solution satisfying the goal situation.

In [**HS96**], Hölldobler and Schneeberger discuss least commitment partial order planning within the declarative setting of the fluent calculus. In this approach, similar to computation of the constraint sets discussed in Section 8.3, if the goal can be reached then the deductive reasoning process yields a partially ordered set of actions.

[**HK00, Leh02**] address decidability issues related to conjunctive planning problems within the fluent calculus. [**HS00**] extends the language of fluent calculus to complex plans including conditional and recursive actions.

The approach in [**MOM99**] offers rewriting logic and its implementation language Maude [**CDE$^+$03**] as a platform for conjunctive and disjunctive planning problems.

Conjunctive planning problems have been studied also from the point of view of linear logic: In [**Jac93**], Jacopin presents an implementation of proof search in multiplicative linear logic with respect to conjunctive planning problems. He addresses two points as drawbacks, namely, the nondeterministic behavior of the context management rules, which requires a lot of backtracking, and the absence of expressing partial goal situations. However, both of these points are due to his representation of the planning problems.

[**KY93b**] presents the linear logic programming language ACL, and applies it to conjunctive planning problems. [**BG94**] discusses the conjunctive planning problems and concurrency in the context of the abstract logic programming language Forum. [**CSR99**] extends the linear logic approach to cover complex and recursive plans. In [**KV01**], Kanovich *et al.* study the complexity of planning problems within Horn linear logic and show that complexity of conjunctive planning problems can be reduced to PSPACE. [**KV03**] discusses a technique for contracting the exponential search space in conjunctive planning problems to a polynomial one by means of abstractions.

CHAPTER 9

Open Problems and Future Work

In this chapter, I collect some problems which follow from the investigations discussed in this thesis and I consider interesting and worthy of further research.

9.1. Reducing Nondeterminism

In Chapter 5, we have seen a proof theoretical technique in the calculus of structures for reducing nondeterminism in proof search. By exploiting an interaction schema on the structures, this technique allows to redesign the inference rules by means of restrictions. The resulting inference rules act on the structures only in those ways that promote the interactions between dual atoms and reduce the interaction between atoms which are not duals of each other. These restrictions on the inference rules reduce the breadth of the search space drastically while preserving the shorter proofs that are available due to deep inference.

We have seen that by replacing the switch rule in system BV with the rule lazy interaction switch, we obtain an equivalent deep inference system where nondeterminism in proof search is reduced in comparison to system BV. The techique that I employed for this purpose exploits a scheme of inference rules which is common to the systems of the calculus of structures without sacrificing from proof theoretical purity. In all the systems of the calculus of structures the switch rule is responsible for the commutative context management.

9.1.1. Reducing Nondeterminism Further in System BV. The rule s and $\mathsf{q}{\downarrow}$ manage the context of the commutative and non-commutative contexts, respectively, in proof construction in a similar way. In fact, Guglielmi obtained these two rules in [**Gug07**] as the instances of the same rule in different contexts. However, the non-commutative context has a quite different behaviour in contrast to the commutative contexts. For instance, on BV structures consider the rule

$$\frac{S([R,U],[T,V])}{S[(R,T),(U,V)]}$$

which is unsound. However, when we replace the copar operators in this instance with the seq operators, we obtain the rule $\mathsf{q}_1{\downarrow}$ which is sound. Because of this, it becomes difficult to carry the ideas on rule s to the rule $\mathsf{q}{\downarrow}$. In particular, the equivalence of systems BV and BVi remain an open problem. In this respect, Conjecture 5.55 remains to be investigated.

As we have seen in Section 5.3, the rule $\mathsf{q}_2{\downarrow}$ is responsible for a great redundant nondeterminism in system BV. However, removing this rule from system BVi results in an incomplete system (see Example 5.58). I believe that redesigning this rule as described in Definition 5.60 would control this redundant nondeterminism in system BV. In this respect, Conjecture 5.62 deserves further investigation.

9.1.2. Nondeterminism in System NEL. System NEL is a conservative extension of system BV. All the rules of system BV are common to system NEL. The splitting technique, which I used to prove the completeness of system BVsl, was used also by Guglielmi and Straßburger in [**GS02**] to prove cut elimination for system NEL in combination with a decomposition theorem. I believe that by combining the ideas in [**GS02**] and in Chapter 5, it should be plausible to reduce nondeterminism in proof search also in system NEL analogously as in system BV.

Apart from the rules s and q↓ which are common to systems BV and NEL, there is another rule in system NEL which has a potential for the application of the technique of this thesis: From the point of view of the notion of interaction that was considered while desinging system BVsl, in the promotion rule

$$\mathsf{p}{\downarrow}\,\frac{S\{![R,T]\}}{S[!R,?T]}$$

the interaction between the structures R and T is stronger in the premise than in the conclusion. By exploiting this observation, I conjecture that this rule can be replaced in, system NEL, with the rule *interaction promotion* which requires R and T to interact to be applied, that is, the interaction promotion can be applied only if $\mathsf{at}\,\bar{R} \cap \mathsf{at}\,T \neq \emptyset$.

9.1.3. Nondeterminism in Other Logics. The technique of this thesis for reducing nondeterminism exploits a scheme which is common to all the systems of the calculus of structures. As we have seen in Chapter 5, this technique can be applied to calculus of structures systems KSg and KS for classical logic. By carrying these ideas to linear logic system LS in combination with the splitting argument in [**Str03a**], it should be possible to obtain deep inference systems for linear logic where nondeterminism is reduced by means of interaction rules. The above mentioned conjecture for the promotion rule in the context of system NEL applies also to the promotion rule in system LS because this rule is common to both systems. Exploring the interaction schema in the additive rules of system LS is another problem that I consider worthy for further investigation.

In [**SS05**], Stewart and Stouppa present calculus of structures systems for a class of modal logics. These systems extend the classical logic system KSg with the modal rules. For instance, a system for modal logic K is obtained by extending system KSg with the following rule:

$$\mathsf{k}{\downarrow}\,\frac{S\{\Box[R,T]\}}{S[\Box R,\Diamond T]}$$

It is important to observe the similarity between this rule and the rule p↓ above. Thus, the conjecture of Subsection 9.1.2 can be carried to this rule. Then, by applying the technique of this thesis for reducing nondeterminism to these systems for modal logics, it should be possible to obtain deep inference systems also for modal logics where nondeterminism is reduced.

9.1.4. Deepest Deep Inference Rules. The interaction rules succeed in reducing nondeterminism in proof search while preserving the shorter proofs that are available due to deep inference. However, when these rules are applied during proof search to structures of arbitrary size, performing the check of the condition of the interaction rules becomes computationally expensive. On the other hand,

when the application of the interaction rules is restricted to the redexes which are deep inside, the check of these conditions must be performed on the "smaller" substructures. This observation gives rise to questions regarding a plausible notion of deeper inference rules.

Deep instances of the inference rules act on the structure at the deeper contexts and this way serve to annihilate the substructures at arbitrary depths. This results in shorter proofs in contrast to the proofs constructed by means of shallower instances of the inference rules. Due to this observation, the idea of giving higher priority to the deeper instances of the inference rules can be used as a search strategy that gives a higher priority to deeper instances of the interaction rules. However, the completeness of the systems which are designed with respect to a plausible notion of deepness remains an interesting research problem. For instance, in [**Str03a**], Straßburger introduces a rule called *deep switch*

$$\mathsf{ds}\,\frac{S([R,U],T)}{S[(R,T),U]}$$

where the structure R is not a proper copar structure. This and other notions of deeper inference rules, in combination with interaction rules, should provide a further proof theoretic reduction in nondeterminism in proof search without losing the shorter proof that were previously available.

9.1.5. The Relationship between System BV and Pomset Logic. There is a strict correspondence between the structures of system BV and the formulae of pomset logic [**Ret97**]. Guglielmi and Straßburger have conjectured in [**Gug07**] and [**Str03a**], respectively, that these logics are equivalent. I consider it worthy to investigate if system BVi provides any simplifications for addressing the equivalence of pomset logic and system BV.

9.1.6. Relationship with the Connection Method. Bibel's *connection method* [**Bib83, Bib87**] is a proof procedure which was originally developed for classical logic. Connection method can be seen as computing complementary connections in matrix representations of logical expressions. In [**KO99**], Kreitz and Otten argue that proofs in connection method can be seen as compact representations of sequent calculus proofs, because, being driven by complementary connections, connection method avoids the usual redundancies contained in the sequent calculus proofs. In other words, connection method focuses on possible leaves in a sequent calculus proof, instead of the logical connectives of the formula being proved. Kreitz and Otten extend the connection method, as a uniform procedure, for proof search in classical logic, intuitionistic logic, a class of modal logics, and multiplicative fragment of linear logic.

Because the calculus of structures is more general than the sequent calculus, the above mentioned observations can be easily carried over from the sequent calculus to the calculus of structures. However, because of the feature of deep inference, the rule $\mathsf{ai}{\downarrow}$ gives a more immediate characterization of the connections of the connection method. In particular, because connection method proofs are rather algorithmic than deductive, and the interaction rules can be applied in only those ways that takes these connections into considerations, the calculus of structures can provide a deductive interpretation for the connection method proofs. In this respect, the relationship between the calculus of structures and the connection

method deserves further investigation. In [**LS05**], Lamarche and Straßburger give a geometric characterization of classical proofs in a way which resembles proof nets of MLL, which, in my opinion, can provide a starting point for these investigations.

9.2. Implementations

9.2.1. Complexity of Proofs. Deep inference feature of the calculus of structures provides shorter proofs than in the sequent calculus for some classes of formulae, as it was shown by Guglielmi in [**Gug04c**]. A thorough comparison of deep inference proofs, also when the technique of Chapter 5 is used, with proofs in the sequent calculus and in other formalisms in terms of proof complexity remains as future work.

9.2.2. Implemeting Different Search Strategies. In the implementations of the calculus of structures systems presented in this thesis, mainly breadth-first search strategy has been employed. Breadth-first search is a complete search strategy. However, for the proofs consisting of more than several steps, proof search by using this strategy results in a search that does not terminate in a plausible amount of time. In particular, for the systems with inference rules that copy structures, e.g., contraction rule, performing breadth-first search without taking the application of such rules under control is not plausible even for very short proofs.

Different search strategies that implement controlled applications of the inference rules, e.g., which copy structures such as contraction, can provide a much better performance in proof search in these systems. Furthermore, randomized search strategies, e.g., random-restart hill climbing [**RN02**], can provide more efficient proof procedures with the price of loss of completeness. Then, by integrating proof theoretic results on these systems in combination with experiments performed with different search strategies can provide more efficient performance in proof search for different classes of structures. Exploiting the meta-level features of the language Maude for implementing such strategies can provide interesting results. The implementation of system DKSg in Subsection 4.3.4 provides an example for the use of meta-level features of Maude in this respect. Similarly, the expressive power of the imperative programming languages can be used to implement different search strategies when the calculus of structures systems are implemented in imperative languages, similar to those presented in Chapter 7.

9.2.3. Automated Proof Manipulation. Apart from the interest in proof search and proof construction, the implementations of the calculus of structures systems can be further developed in a way to accommodate proof manipulation facilities. Cut elimination results in the calculus of structures systems, e.g., in [**Gug07, Str03a, Brü03b**], as well as permutation of the inference rules and transformation of derivations into derivations in normal forms, e.g., decomposition results in [**Str03a**] and [**Brü03b**], can be automatized by means of implementations. Along these lines, in [**Sch06**], Schäfer extends the Maude implementations of this thesis by defining a data structure for representing derivations. He then provides a proof manipulation functionality that replaces the rule instances in a derivation with derivations with the same premise and conclusion as the replaced rule instance. This way, he implements the automated elimination of the rules $\mathsf{c}\uparrow$ in system SKS [**Brü03b**] for classical logic as the first step of the cut-elimination procedure which modularly eliminates the up rules of system SKS. In my opinion,

the work by Schäfer provides the basis for implementing more complex proof manipulation tools which can perform tasks such as cut-elimination, permutation of rules in a derivation, or transforming a given derivation into a normal form.

Guglielmi defines two formalisms, called *formalism A* [**Gug04a**] and *formalism B* [**Gug04b**], that are more general than the calculus of structures, where the concurrency in the proofs can be represented explicitly at different syntactic levels as parallel derivations. Automatic transformations from the derivations in the calculus of structures into derivations in formalism A and formalism B is another potential direction for further developing the implementations of this thesis.

9.2.4. Implementing Systems with Quantifiers. In contrast to the propositional systems, systems that involve quantifiers is a less studied topic in the context of deep inference so far [1]. In [**Brü06**], Brünnler gives a deep inference system for classical predicate logic together with a cut-elimination proof. In [**Str03a**], Straßburger describes potential quantifier rules which can be used to extend systems for linear logic, leaving their proof theoretic study as an open problem.

In this thesis, I have discussed only the implementation of propositional deep inference systems, leaving out the treatment of the quantifiers in deep inference systems. A possible direction to proceed along these lines is by means of *explicit substitutions* [**ACCL91**]. In [**MOM96**], Marti-Oliét and Meseguer give encodings of the sequent calculus systems with quantifiers by resorting to explicit substitutions in an older version of the language Maude than the one I employed in this thesis. It remains to investigate if the methods of [**MOM96**] can be used to implement the deep inference systems with quantifiers in Maude.

9.3. Language Design

In Chapter 8, I have introduced a common language for planning and concurrency, called $\mathcal{K}$. In language $\mathcal{K}$, the sequential and parallel composition of actions can be expressed at the same logical level, and this way logical reasoning can be performed on these plans. In Section 8.4, I have introduced a notion of plan equivalence which can be used to verify if two plans can be replaced with each other in a given context. The equivalence of two such plans with respect to this notion can then be checked by proof search in the systems being used.

9.3.1. Equivalence of Plans. In [**Leh02**], Lehmann studies the planning problems of this thesis as interleaving processes to study their bisimilarity in order to address decidability issues related to fluent calculus. A natural question to ask is if a plan equivalence result based on bisimilarity can be established for language $\mathcal{K}$ similar to the one in [**Leh02**]. Carrying over results from the study of labeled event structures or petri nets for a better understanding of the planning problems in the context of language $\mathcal{K}$ is another direction of research.

9.3.2. Verification of Security Protocols. The relationship between multiset rewriting and verification of security protocols with respect to the Dolev-Yao model [**DY83**] has been studied by various authors (see, e.g., [**DLJS04, BCLM05**]). In such a context, verification of a security protocol can be easily put as a planning problem: "Is there a sequence of actions that an intruder can

[1] `http://alessio.guglielmi.name/res/cos/`

undertake so that he can break a security protocol?". The plausibility of language $\mathcal{K}$ for such concurrency theoretic queries, and others, remains to be investigated.

Bibliography

[Abr91] Vito Michele Abrusci, *Phase semantics and sequent calculus for pure non-commutative classical linear logic*, Journal of Symbolic Logic **56(4)** (1991), 1403–1451.

[Abr93] Samson Abramsky, *Computational interpretations of linear logic*, Theoretical Computer Science **111** (1993), 3–57.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy, *Explicit substitution*, Journal of Functional Programming **1(4)** (1991), 375–416.

[AK01] José Luis Ambite and Craig A. Knoblock, *Planning by rewriting*, Journal of Artificial Intelligence Research **15** (2001), 207–261.

[Ale94] Vladimir Alexiev, *Applications of linear logic to computation: An overview*, Bulletin of IGPL **2** (1994), no. 1, 77–107, Technical Report at the University of Alberta, TR93–18.

[And92] Jean-Marc Andreoli, *Logic programming with focussing proofs in linear logic*, Journal of Logic and Compututation **2(3)** (1992), 297–347.

[And01] ———, *Focussing and proof construction*, Annals of Pure and Applied Logic **107** (2001), 131–163.

[AP91] Jean-Marc Andreoli and Remo Pareschi, *Linear ojects: Logical processes with built-in inheritance.*, New Generation Computing **9** (1991), no. 3/4, 445–474.

[AR00] Vito Michele Abrusci and Paul Ruet, *Non-commutative logic: The multiplicative fragment*, Annals of Pure and Applied Logic **101** (2000), 29–64.

[Asp87] A. Asperti, *A logic for concurrency*, Tech. report, Universita di Pisa, 1987.

[Bäc98] C. Bäckström, *Computational aspects of reordering plans*, Journal of Artificial Intelligence Research **9** (1998), 99–137.

[Bae04] J.C.M. Baeten, *A brief history of process algebra*, Tech. Report Rapport CSR 04-02, Vakgroep Informatica, Technische Universiteit Eindhoven, 2004.

[BB00] J.C.M. Baeten and T. Basten, *Partial-order process algebra (and its relation to petri nets)*, Handbook of Process Algebra (J.A. Bergstra, A. Ponse, and S.A.Smolka, eds.), Elsevier, 2000, pp. 769–872.

[BB01] C. Boutilier and R. Brafman, *Partial-order planning with concurrent interacting actions*, Journal of Artificial Intelligence Research **14** (2001), 105–136.

[BCLM05] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli, *Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis*, Journal of Computer Security **13** (2005), no. 1, 3–47.

[Bel82] Nuel D. Belnap, *Display logic*, Journal of Philosophical Logic **11** (1982), 375–417.

[Bel97] G. Bellin, *Subnets of proof-nets in multiplicative linear logic with MIX*, Mathematical Structures in Computer Science, vol. 7, Cambridge University Press, 1997, pp. 663–699.

[BG94] Paola Bruscoli and Alessio Guglielmi, *Expressiveness of the abstract logic programming langauge Forum in planning and concurrency*, Proccedings of the Joint Conference on Declarative Programming,GULP-PRODE 94 (Spain), vol. 2, 1994, pp. 221–237.

[BG97] Chitta Baral and Michael Gelfond, *Reasoning about effects of concurrent actions*, Journal of Logic Programming **31** (1997), no. 1-3, 85–117.

[BG03] Paola Bruscoli and Alessio Guglielmi, *A tutorial on proof theoretic foundations of logic programming*, Proceedings of the 19th International Conference on Logic Programming, ICLP 2003, Lecture Notes in Computer Science, vol. 2916, Springer-Verlag, 2003, pp. 109–127.

[BHS+93] S. Brüning, S. Hölldobler, U. C. Sigmund, M. Thielscher, and J. Schneeberger, *Disjunction in resource-oriented deductive planning*, Proceedings of the 1993 international symposium on Logic programming (Vancouver, British Columbia, Canada), 1993, pp. 670–675.

[Bib83] Wolfgang Bibel, *Matings in matrices*, Communications of the ACM (CACM) **26(11)** (1983), 844–852.

[Bib86] ———, *A deductive solution for plan generation*, New Generation Computing **4** (1986), 115–132.

[Bib87] Wolfgang Bibel, *Automated theorem proving*, Vieweg Verlag, 1987.

[BKK+98] P. Borovansky, C. Kirchner, H. Kirchner, P. Moreau, and C. Ringeissen, *An overview of elan*, Proceedings of the second International Workshop on Rewriting Logic and its Applications (C. Kirchner and H. Kirchner, eds.), Electronic Notes in Theoretical Computer Science, vol. 15, 1998.

[BM97] A. Blum and Furst M., *Fast planning through planning graph analysis"*, Artificial Intelligence **90** (1997), 281–300.

[BN98] Franz Baader and Tobias Nipkow, *Term rewriting and all that*, vol. 1, Cambridge University Press, 1998.

[BP98] Paul Beame and Toniann Pitassi, *Propositional proof complexity: Past, present, and future*, Bulletin of the European Association for Theoretical Computer Science **65** (1998), 66–89.

[BPS01] Jan A. Bergstra, Alban Ponse, and Scott A. Smolka (eds.), *Handbook of process algebra*, Elsevier, 2001.

[Bre91] Gerhard Brewka, *Nonmonotonic reasoning: Logical foundations of commonsense*, Cambridge Tracts in Theoretical Computer Science, vol. 12, Cambridge University Press, 1991.

[Bru02] Paola Bruscoli, *A purely logical account of sequentiality in proof search*, Logic Programming, 18th International Conference (Peter J. Stuckey, ed.), Lecture Notes in Computer Science, vol. 2401, Springer-Verlag, 2002, pp. 302–316.

[Brü03a] Kai Brünnler, *Atomic cut elimination for classical logic*, CSL 2003 (M. Baaz and J. A. Makowsky, eds.), Lecture Notes in Computer Science, vol. 2803, Springer-Verlag, 2003, pp. 86–97.

[Brü03b] ———, *Deep inference and symmetry in classical proofs*, Ph.D. thesis, Technische Universität Dresden, 2003.

[Brü06] ———, *Cut elimination inside a deep inference system for classical predicate logic*, Studia Logica **82** (2006), no. 1, 51–71.

[BS96] Sven-Erik Bornscheuer and Torsten Seiler, *Massively parallel reasoning about actions*, KI - Kunstliche Intelligenz, 1996, pp. 5–17.

[BT01] Kai Brünnler and Alwen Fernanto Tiu, *A local system for classical logic*, LPAR 2001 (R. Nieuwenhuis and A. Voronkov, eds.), Lecture Notes in Artificial Intelligence, vol. 2250, Springer-Verlag, 2001, pp. 347–361.

[Bus98] Samuel R. Buss, *An introduction to proof theory*, Handbook of Proof Theory (S. R. Buss, ed.), Elsevier, 1998, pp. 1–78.

[Byl92] T. Bylander, *Complexity results for serial decomposability*, Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92) (San Jose), AAAI Press, 1992, pp. 729–734.

[CDE+99] Manuel Clavel, Francisco Durán, Steven Eker, José Meseguer, and Mark-Oliver Stehr, *Maude as a formal meta-tool*, FM'99 — Formal Methods, World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 20–24, 1999 Proceedings, Volume II (Jeannette M. Wing, Jim Woodcock, and Jim Davies, eds.), Lecture Notes in Computer Science, vol. 1709, Springer, 1999, pp. 1684–1703.

[CDE+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada, *Maude: specification and programming in rewriting logic*, Theoretical Computer Science **285** (2002), 187–243.

[CDE+03] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *The Maude 2.0 system*, Rewriting Techniques and Applications, Proceedings of the 14th International Conference (Robert Nieuwenhuis, ed.), vol. 2706, Springer, 2003.

[CDE+05] ———, *Maude 2.1.1 manual*, Tech. report, Computer Science Laboratory, SRI International, 2005, http://maude.cs.uiuc.edu/manual/.

[Cer95] Iliano Cervesato, *Petri nets and linear logic: a case study for logic programming*, Proceedings of the Joint Conference on Declarative Programming: GULP-PRODE'95 (Marina di Vietri, Ital), 1995.

[CHP96] Iliano Cervesato, Joshua Hodas, and Frank Pfenning, *Efficient research management for linear logic proof search*, Proceedings of the 1996 Workshop on Extensions to Logic Programming (Leipzig, Germany) (R. Dyckhoff, H. Herre, and P. Schroeder-Heister, eds.), Lecture Notes in Artificial Intelligence, vol. 1050, 1996, pp. 28–30.

[Cla00] Manuel Clavel, *Reflection in rewriting logic: Metalogical foundations and metaprogramming applications*, CSLI Publications, 2000.

[CM96] Manuel Clavel and José Meseguer, *Reflection and strategies in rewriting logic*, Proceedings First International Workshop on Rewriting Logic and its Applications, WRLA'96, Asilomar, California (José Meseguer, ed.), Electronic Notes in Theoretical Computer Science, vol. 4, Elsevier, 1996, pp. 125–147.

[CSR99] S. Cresswell, A. Smaill, and J. Richardson, *Deductive synthesis of recursive plans in linear logic*, ECP, 1999, pp. 252–264.

[Dep00] Eric Deplagne, *Sequent calculus viewed modulo*, Proceedings of the ESSLLI-2000 Student Session (Birmingham, England) (Catherine Pilière, ed.), University of Birmingham, August 2000, pp. 66–76.

[Dep02] ———, *Système de preuve modulo récurrence*, Thèse de doctorat, Université Nancy 1, 2002.

[DGC96] Giuseppe De Giacomo and XiaoJun Chen, *Reasoning about nondeterministic and concurrent actions: a process algebra approach*, Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), vol. 1, 1996.

[DGLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque, *ConGolog, a concurrent programming language based on the situation calculus*, Artificial Intelligence **121 (1-2)** (2000), 109–169.

[DLJS04] N.A Durgin, P.D. Lincoln, J.C.Mitchell, and A. Scedrov, *Multiset rewriting and the complexity of bounded security protocols*, Journal of Computer Security **12 (2)** (2004), 247–311.

[DQ03] R.J. De Queiroz (ed.), *Logic for concurrency and synchronisation*, Trends in Logic – Studia Logica Library, vol. 18, Kluwer Academic Publishers, 2003.

[Dru89] Mark Drummond, *Situated control rules*, Proceedings of the first international conference on Principles of knowledge representation and reasoning (San Francisco, CA, USA), Morgan Kaufmann, 1989, pp. 103–113.

[DY83] D. Dolev and A. Yao, *On the security of public-key protocols*, IEEE Transactions on Information Theory **2(29)** (1983), 198–208.

[EHT96] Kerstin Eder, Steffen Hölldobler, and Michael Thielscher, *An abstract machine for reasoning about situations, actions, and causality*, 5th International Workshop on Extensions of Logic Programming, Lecture Notes in Artificial Intelligence 1050, Springer Verlag, March 1996, pp. 137–151.

[EW93] U. Engberg and G. Winskel, *Completeness results for linear logic on Petri nets*, Proceedings of the Conference on Mathematical Foundations of Computer Science (Gdańsk, Poland) (A. Borzyszkowski and S. Sokolowski, eds.), Lecture Notes in Computer Science, vol. 711, Springer-Verlag, 1993, pp. 442–452.

[EW94] ———, *Linear logic on petri nets*, In A Decade of Concurrency: Reflections and Perspectives, REX School/Symposium Proceedings, Lecture Notes in Computer Science, vol. 803, Springer Verlag, 1994, In BRICS reports series RS-94-3.

[FN71] R. E. Fikes and H. J. Nilsson, *STRIPS: A new approach to the application of theorem proving to problem solving*, Artificial Intelligence **2** (1971), 189–205.

[FR94] Arnaud Fleury and Christian Retoré, *The mix rule*, Mathematical Structures in Computer Science **4(2)** (1994), 273–285.

[Gb01] Alessio Guglielmi and Lutz Straßburger, *Non-commutativity and MELL in the calculus of structures*, CSL 2001 (L. Fribourg, ed.), Lecture Notes in Computer Science, vol. 2142, Springer-Verlag, 2001, pp. 54–68.

[Gen34] Gerhard Gentzen, *Untersuchungen über das logische Schließen I*, Mathematische Zeitschrift **39** (1934), 176–210.

[Gen35] ———, *Untersuchungen über das logische Schließen II*, Mathematische Zeitschrift **39** (1935), 405–431.

[Gen69] ———, *Investigations into logical deduction*, The Collected Papers of Gerhard Gentzen (M.E. Szabo, ed.), NorthHolland Publishing Co., Amsterdam, 1969, pp. 68–131.

[GG89] Carl Gunter and Vijay Gehlot, *Nets as tensor theories*, Tech. Report MS-CIS-89-68, University of Pennsylvania, October 1989.

[GHS96] G. Große, S. Hölldobler, and J. Schneeberger, *Linear deductive planning*, Journal of Logic and Computation **6 (2)** (1996), 233–262.

[Gir87] Jean-Yves Girard, *Linear logic*, Theoretical Computer Science **50** (1987), 1–102.

[GJ79] M. R. Garey and D. S. Johnson, *Computer and intractability: A guide to theory of NP-completeness*, W. H. Freeman & Co., San Francisco, 1979.

[GL93] Michael Gelfond and Vladimir Lifschitz, *Representing action and change by logic programs*, Journal of Logic Programming **17** (1993), no. 2/3–4, 301–321.

[GL98] ———, *Action languages*, Electronic Transactions on Artificial Intelligence **2 (3–4)** (1998), 193–210.

[GLKM98] M. Ghallab, A. Lowe, C. A. Knoblock, and D. McDermott, *PDDL-the planning domain definition language*, Tech. Report DCS TR-1165, Yale Center for Computational Vision and Control, New Heaven, Connecticut, 1998.

[GS02] Alessio Guglielmi and Lutz Straßburger, *A non-commutative extension of MELL*, LPAR 2002 (M. Baaz and A. Voronkov, eds.), Lecture Notes in Artificial Intelligence, vol. 2514, Springer-Verlag, 2002, pp. 231–246.

[Gue99] Stefano Guerrini, *Correctness of multiplicative proof nets is linear*, Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society, 1999, pp. 454–463.

[Gug96] Alessio Guglielmi, *Abstract logic programming in linear logic independence and causality in a first order calculus*, Ph.D. thesis, Universita di Pisa – Genova, 1996.

[Gug02] ———, *Recipe*, Manuscript. Available on the web at http://cs.bath.ac.uk/ag/p/AG2.pdf, 2002.

[Gug03] ———, *Mismatch*, Available on the web at http://cs.bath.ac.uk/ag/p/AG9.pdf, 2003.

[Gug04a] ———, *Formalism a*, Manuscript. Available on the web at http://cs.bath.ac.uk/ag/p/AG11.pdf, 2004.

[Gug04b] ———, *Formalism b*, Manuscript. Available on the web at http://cs.bath.ac.uk/ag/p/AG13.pdf, 2004.

[Gug04c] ———, *Polynomial size deep-inference proofs instead of exponential size shallow-inference proofs*, Available on the web at http://cs.bath.ac.uk/ag/p/AG12.pdf, 2004.

[Gug04d] ———, *Splitting for classical logic*, Email to Frogs mailing-list. Available on the web at http://article.gmane.org/gmane.science.mathematics.frogs/134/match=splitting, 2004.

[Gug07] ———, *A system of interaction and structure*, ACM Transactions on Computational Logic **8** (2007), no. 1, 1–64.

[HK00] Steffen Hölldobler and Dietrich Kuske, *The boundary between decidable and undecidable fragments of the fluent calculus*, International Conference on Logic for Programming and Automated Reasoning 2000, Lecture Notes in Computer Science, vol. 1955, Springer, 2000, pp. 436–450.

[HK04] Steffen Hölldobler and Ozan Kahramanoğulları, *From the calculus of structures to term rewriting systems*, Tech. Report WV-04-03, TU Dresden, 2004.

[HM85] M. Hennesy and R. Milner, *Algebraic laws for nondeterminism and concurrency*, Journal of ACM **32** (1985), 137–161.

[HM94] Joshua Hodas and Dale Miller, *Logic programming in a fragment of intuitionistic linear logic*, Information and Computation **110 (2)** (1994), 327–365.

[Hod94] Joshua S. Hodas, *Logic programming in intuitionistic linear logic: Theory, design, and implementation*, Ph.D. thesis, University of Pennsylvania, 1994.

[Hof03] J. Hoffmann, *Utilizing problem structure in planning: A local search approach*, Lecture Notes in Artificial Intelligence, vol. 2854, Springer Verlag, 2003.

[How80] W. A. Howard, *The formulae-as-types notion of construction*, To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism (J. P. Seldin and J. R. Hindley, eds.), Academic Press, 1980, pp. 479–490.

[HP03] James Harland and David Pym, *Resource-distribution via boolean constraints*, Transations on Computational Logic **4 (1)** (2003), 56–90.

[HS90] Steffen Hölldobler and Josef Schneeberger, *A new deductive approach to planning*, New Generation Computing (1990), 225–244.

[HS96] Steffen Hölldobler and Josef Schneeberger, *Constraint equational logic programming and resource-based partial order planning*, Tech. Report WV-96-08, TU Dresden, 1996.

[HS00] Steffen Hölldobler and Hans-Peter Störr, *Complex plans in the fluent calculus*, Intellectics and Computational Logic: Papers in Honor of Wolfgang Bibel (Steffen Hölldobler, ed.), Kluwer Academic Press, 2000, pp. 207–223.

[HS05] Robert Hein and Charles Stewart, *Purity through unravelling*, Structures and Deduction (Paola Bruscoli, François Lamarche, and Charles Stewart, eds.), Technische Universität Dresden, 2005, ISSN 1430-211X, pp. 126–143.

[HT93] Steffen Hölldobler and Michael Thielscher, *Actions and specificity*, Proceedings of the International Logic Programming Symposium (ILPS) (Vancouver) (D. Miller, ed.), MIT Press, 1993, pp. 164–180.

[IH01] K. Ishihara and K. Hiraishi, *The completeness of linear logic for petri net models*, Logic Journal of IGPL **9(4)** (2001), 549–567.

[Jac93] Éric Jacopin, *Classical AI planning as theorem proving: The case of a fragment of linear logic*, AAAI Fall Symposium on Automated Deduction in Nonstandard Logics (Palo Alto, California), AAAI Press Publications, 1993, pp. 62–66.

[Kah04a] Ozan Kahramanoğulları, *Implementing system BV of the calculus of structures in Maude*, Proceedings of the ESSLLI-2004 Student Session (Université Henri Poincaré, Nancy, France) (Laura Alonso i Alemany and Paul Égré, eds.), 2004, 16th European Summer School in Logic, Language and Information, pp. 117–127.

[Kah04b] Ozan Kahramanoğulları, *System BV without the equalities for unit*, Proceedings of the 19th International Symposium on Computer and Information Sciences, ISCIS'04 (Cevdet Aykanat, Tuğrul Dayar, and Ibrahim Körpeoğlu, eds.), Lecture Notes in Computer Science, vol. 3280, Springer, 2004.

[Kah05] Ozan Kahramanoğulları, *Towards planning as concurrency*, Proceedings of the The IASTED International Conference on Artificial Intellgence and Applications, AIA 2005 (Innsbruck, Austria), 2005, pp. 387–393.

[Kah06a] Ozan Kahramanoğulları, *Reducing nondeterminism in the calculus of structures*, LPAR 2006 (M. Hermann and A. Voronkov, eds.), Lecture Notes in Artificial Intelligence, vol. 4246, Springer-Verlag, 2006, pp. 272–286.

[Kah06b] Ozan Kahramanoğulları, *System BV is NP-complete*, Proceedings of the 12th Workshop on Logic, Language, Information and Computation (WoLLIC 2005), July 2005 (Florianapolis, Brazil) (R. de Queiroz, A. Macintyre, and G. Bittencourt, eds.), Electronic Notes in Theoretical Computer Science, vol. 143, Elsevier, 2006, pp. 87–99.

[Kah07] Ozan Kahramanoğulları, *System BV is NP-complete*, Annals of Pure and Applied Logic (2007), to appear.

[Kan91] Max Kanovich, *The multiplicative fragment of linear logic is NP-complete*, Tech. Report X-91-13, Institute for Language, Logic, and Information, 1991.

[Kan92] ———, *Horn programming in linear logic is NP-complete*, Proccedings of the 7th Annual IEEE Symposium on Logic in Computer Science, Santa Cruz (Los Alamitos, California), IEEE Computer Society Press, 1992, pp. 200–210.

[KG99] Lars Karlsson and Joakim Gustafsson, *Reasoning about concurrent interaction*, Journal of Logic and Computation **9(5)** (1999), 623–650.

[KK04] Claude Kirchner and Hélène Kirchner, *Rule-based programming and proving: The ELAN experience outcomes*, Advances in Computer Science - ASIAN 2004, Proccedings of Ninth Asian Computing Science Conference (Chiang Mai, Thailand) (Michael J. Maher, ed.), Lecture Notes in Computer Science, vol. 3321, Springer, 2004, p. 363.

[KMR05a] Ozan Kahramanoğulları, Pierre-Etienne Moreau, and Antoine Reilles, *Implementing deep inference in TOM*, Structures and Deduction – the Quest for the Essence of

Proofs (satellite workshop of ICALP 2005), Technical Report ISSN 1430-211X, Technische Universität Dresden (Lisbon, Portugal) (Paola Bruscoli, François Lamarche, and Charles Stewart, eds.), 2005, pp. 158–172.

[KMR05b] Claude Kirchner, Pierre-Etienne Moreau, and Antoine Reilles, *Formal validation of pattern matching code*, Proceedings of the 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, ACM Press, 2005, pp. 187 – 197.

[Kno94] Craig A. Knoblock, *Generating parallel execution plans with a partial-order planner*, Artificial Intelligence Planning Systems, 1994, pp. 98–103.

[KO99] Christoph Kreitz and Jens Otten, *Connection-based theorem proving in classical and non-classical logics*, Journal of Universal Computer Science **5(3)** (1999), 88–112.

[KS98] H. Kautz and B. Selman, *Blackbox: A new approach to the application of theorem proving to problem solving*, Working Notes of the AIPS-98 Workshop on Planning as Combinatorial Search, 1998.

[KT03] Ozan Kahramanoğulları and Michael Thielscher, *A formal assessment result for fluent calculus using the action description language* $\mathcal{A}_k$, Proceedings of the German Annual Conference on Artificial Intelligence (KI) (Hamburg, Germany) (R. Kruse, ed.), Lecture Notes in Artificial Intelligence, vol. 2821, Springer, 2003, pp. 209–223.

[Kün03] Peep Küngas, *Linear logic for domain-independent ai planning (extended abstract)*, Proceedings of Doctoral Consortium at 13th International Conference on Automated Planning and Scheduling, ICAPS 2003 (Trento, Italy), 2003, pp. 68–72.

[Kün05] ———, *Petri net reachability checking is polynomial with optimal abstraction hierarchies*, Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation, SARA 2005 (Airth Castle, Scotland, UK), Lecture Notes in Artificial Intelligence, vol. 3607, Springer-Verlag, 2005, pp. 149–164.

[KV01] M. I. Kanovich and J. Vauzeilles, *The classical AI planning problems in the mirror of horn linear logic: semantics, expressibility, complexity*, Mathematical Structures in Computer Science **11(6)** (2001), 689–716.

[KV03] M. I. Kanovich and J. Vauzeilles., *Coping polynomially with numerous but identical elements within planning problems*, CSL 2003 (M. Baaz and J. A. Makowsky, eds.), Lecture Notes in Computer Science, vol. 2803, Springer-Verlag, 2003, pp. 285–298.

[KY93a] N. Kobayashi and A. Yonezawa, *ACL — A concurrent linear logic programming paradigm*, Proceedings of the 1993 International Logic Programming Symposium (Vancouver, Canada) (D. Miller, ed.), MIT Press, 1993, pp. 279–294.

[KY93b] ———, *Reasoning on actions and change in linear logic programming*, Tech. report, Department of Information Science, University of Tokyo, 1993.

[Lam58] Joachim Lambek, *The mathematics of sentence structure*, American Mathematical Monthly **65** (1958), 154–169.

[Leh02] Helko Lehmann, *On reasoning about action and change in the fluent calculus*, Ph.D. thesis, Technische Universitat Dresden, Fakultat Informatik, August 2002.

[Lif86] Vladimir Lifschitz, *On the semantics of STRIPS*, Reasoning about Actions and Plans: Proceedings of the 1986 Workshop (Timberline, Oregon) (Michael P. Georgeff and Amy L. Lansky, eds.), Morgan Kaufmann, June-July 1986, pp. 1–9.

[Lip76] R. J. Lipton, *The reachability problem requires exponential space*, Tech. Report 62, Yale University, 1976.

[Llo87] J.W. Lloyd, *Foundations of logic programming*, Springer Verlag, 1987.

[LMSS90] Patrick Lincoln, John Mitchell, Andre Scedrov, and Natarajan Shankar, *Decision problems for propositional linear logic*, Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, FOCS'90 (St Louis, Missouri, USA), IEEE Computer Society Press, 1990, pp. 662–671.

[LRL+97] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl, *GOLOG: A logic programming language for dynamic domains*, Journal of Logic Programming **31,1-3** (1997), 59–83.

[LS92] P. Lincoln and V. Saraswat, *Higher-order, linear, concurrent constraint programming*, Tech. report, Xerox Palo Alto Research Center, 1992.

[LS05] François Lamarche and Lutz Straßburger, *Naming proofs in classical propositional logic*, Typed Lambda Calculi and Applications, TLCA 2005 (Paweł Urzyczyn, ed.), Lecture Notes in Computer Science, vol. 3461, Springer-Verlag, 2005, pp. 246–261.

[LT01] Jorge Lobo and Gisela Mendezand Stuart R. Taylor, *Knowledge and the action description language a*, Theory and Practice of Logic Programming **(2) 1** (2001), 129–184.

[LW94] Patrick Lincoln and Timothy C. Winkler, *Constant-only multiplicative linear logic is NP-complete*, Theoretical Computer Science **135(1)** (1994), 155–169.

[McC59] John McCarthy, *Programs with common sense*, Mechanisation of Thought Processes, Proceedings of the National Physics Laboratory (London, UK), Her Majesty's Stationery Office, 1959, pp. 77–84.

[McC63] ———, *Situations, actions and causal laws*, Tech. Report Memo 2, Stanford University Artificial Intelligence Laboratory, 1963.

[McC86] ———, *Applications of circumscription to formalizing common sense knowledge*, Artificial Intelligence **28** (1986), 89–116.

[Mes92] José Meseguer, *Conditional rewrite logic as a unified model of concurrency*, Theoretical Computer Science **96(1)** (1992), 73–155.

[Mes98] J. Meseguer, *Membership algebra as a logical framework for equational specification*, In 12th International Workshop on Recent Trends in Algebraic Development Techniques (WADT'97), Lecture Notes in Computer Science, vol. 1376, Springer-Verlag, 1998, pp. 18–61.

[MH69] J. McCarthy and P. J Hayes, *Some philosophical problems from the standpoint of ai*, Machine Intelligence (Meltzer B. and Michie D., eds.), vol. 4, Edinburgh University Press, 1969, pp. 463–501.

[Mil89] Robin Milner, *Communication and concurrency*, International Series in Computer Science, Prentice Hall, 1989.

[Mil92] Dale Miller, *The π-calculus as a theory in linear logic: Preliminary results*, Proceedings of the 1992 Workshop on Extensions to Logic Programming (E. Lamma and P. Mello, eds.), Lecture Notes in Computer Science, no. 660, Springer-Verlag, 1992, pp. 242–265.

[Mil95] ———, *Lambda prolog: An introduction to the language and its logic*, Draft of book, 1995.

[Mil96] ———, *Forum: A multiple-conclusion specification logic*, Theoretical Computer Science **165** (1996), 201–232.

[Mil04] ———, *Overview of linear logic programming*, Linear Logic in Computer Science (Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Phil Scott, eds.), vol. 316, Cambridge University Press, 2004, London Mathematical Society Lecture Note.

[MN86] Dale Miller and Gopalan Nadathur, *Higher-order logic programming*, Proceedings of the Third International Logic Programming Conference (Ehud Shapiro, ed.), 1986, pp. 448–462.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov, *Uniform proofs as a foundation for logic programming*, Annals of Pure and Applied Logic **51** (1991), 125–157.

[MOM91] N. Martí-Oliet and J. Meseguer, *From petri nets to linear logic*, Mathematical Structures in Computer Science **1** (1991), 66–101.

[MOM96] ———, *Rewriting logic as a logical and semantic framework*, Proc. 1st Internat Workshop on Rewriting Logic and its Application, WRLA' 96 (J. Meseguer, ed.), Electronic Notes in Theoretical Computer Science, vol. 4, Elsevier, 1996, pp. 189–224.

[MOM99] Narciso Mart'i-Oliet and José Meseguer, *Action and change in rewriting logic*, Dynamic Worlds: From the Frame Problem to Knowledge Management (R. Pareschi and B. Fronhofer, eds.), vol. 11–2, Kluwer Academic Publishers, 1999, pp. 1–53.

[MOM02] N. Martí-Oliet and J. Meseguer, *Rewriting logic: Roadmap and bibliography*, Theoretical Computer Science **285** (2002), no. 2, 121–154.

[MRV03] Pierre-Etienne Moreau, Christophe Ringeissen, and Marian Vittek, *A Pattern Matching Compiler for Multiple Target Languages*, 12th Conference on Compiler Construction, Warsaw (Poland) (G. Hedin, ed.), Lecture Notes in Computer Science, vol. 2622, Springer-Verlag, May 2003, pp. 61–76.

[MTV90] M. Masseron, C. Tollu, and J. Vauzeilles, *Generating plans in linear logic I–II*, Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 472, Springer-Verlag, 1990, pp. 63–75.

[Ndj01] Gilbert Ndjatou, *Minimizing agent specifications using a logic of knowledge and actions*, Journal of Logic and Computation **11–2** (2001), 337–354.

[NK01] X. Nguyen and S. Kambhampati, *Revising partial order planning*, Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01) (Seattle), Morgan Kaufmann, 2001, pp. 459–466.

[NM90] Gopalan Nadathur and Dale Miller, *Higher-order horn clauses*, Journal of the ACM **37(4)** (1990), 777–814.

[Ped89] E. P. D. Pednault, *ADL: Exploring the middle ground between STRIPS and the situation calculus*, Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference (KR-89) (Toronto, ON) (R. Brachmann, H. J. Levesque, and R. Reiter, eds.), Morgan Kaufmann, 1989, pp. 324–332.

[Pet62] C. A. Petri, *Kommunikation mit automaten*, Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.

[Pin94] Javier Pinto, *Temporal reasoning in the situation calculus*, Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, Canada, January 1994.

[Pla93] David A. Plaisted, *Equational reasoning and term rewriting systems*, The Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 1: Deductive Methodologies (Dov Gabbay, Christopher Hogger, and J. A. Robinson, eds.), Oxford University Press, Oxford, 1993, pp. 274–367.

[PPM96] David Pym, Louise Pryor, and David Murphy, *A note on processes for plan-execution and powerdomains for plan-comparison*, Tech. report, Department of Computer Science, Queen Mary and Westfield College, University of London, 1996.

[PS81] Gerald E. Peterson and Mark E. Stickel, *Complete sets of reductions for some equational theories*, Journal of ACM **28** (1981), 223–264.

[PW92] J. Penberthy and D. Weld, *Ucpop: A sound, complete, partial order planner for adl*, KR 92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, 1992, pp. 103–114.

[Rei91] Raymond Reiter, *The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression*, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, Academic Press, 1991, pp. 359–380.

[Rei96] R. Reiter, *Natural actions, concurrency and continuous time in the situation calculus.*, Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, Morgan Kaufmann, 1996, pp. 2–13.

[Rei01] Raymond Reiter, *Knowledge in action*, M.I.T. Press, 2001.

[Ret93] Christian Retoré, *Réseaux et séquents ordonnés*, Ph.D. thesis, Université Paris 7, 1993.

[Ret97] ———, *Pomset logic: A non-commutative extension of classical linear logic*, Typed Lambda Calculus and Applications, TLCA'97 (Ph. de Groote and J. R. Hindley, eds.), Lecture Notes in Computer Science, vol. 1210, Springer-Verlag, 1997, pp. 300–318.

[Ret99] ———, *Pomset logic as a calculus of directed cographs*, Dynamic Perspectives in Logic and Linguistics (Bulzoni, Rome) (V. M. Abrusci and C. Casadio, eds.), 1999, Also available as INRIA Rapport de Recherche RR-3714, pp. 221–247.

[RN02] Stuart J. Russell and Peter Norvig, *Artificial intelligence: A modern approach (2nd edition)*, Prentice Hall, 2002.

[Rob00] John Alan Robinson, *Computational logic: Memories of the past and challenges for the future*, Proccedings of the Computational Logic - CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings (John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, eds.), Lecture Notes in Computer Science, vol. 1861, Springer, 2000, pp. 1–24.

[Rue00] Paul Ruet, *Non-commutative logic II: Sequent calculus and phase semantics*, Mathematical Structures in Computer Science **10** (2000), 277–312.

[Sch06] Max Schäfer, *The design and implementation of the grape graphical proof editor*, Master's Project Report, TU Dresden. Available at http://grape.sourceforge.net/grape.pdf, 2006.

[Sha97] Murray Shanahan, *Solving the frame problem, a mathematical investigation of the common sense law of inertia*, M.I.T. Press, 1997.

[Sha99] ———, *The event calculus explained*, Artificial Intelligence Today (Berlin) (M. J. Wooldridge and M. Veloso, eds.), Lecture Notes in Computer Science, vol. 1600, Springer Verlag, 1999, pp. 409–430.

[Sin98] Munindar P. Singh, *Applying the mu-calculus in planning and reasoning about action*, Journal of Logic and Computation, vol. 8, Oxford University Press, 1998, pp. 425–445.

[SNW96] Vladimiro Sassone, Morgens Nielsen, and Glynn Winskel, *Models for concurrency: Towards a classification*, Theoretical Computer Science **170 (1–2)** (1996), 297–348.

[SS05] Charles Stewart and Phiniki Stouppa, *A systematic proof theory for several modal logics*, Advances in Modal Logic (Renate Schmidt, Ian Pratt-Hartmann, Mark Reynolds, and Heinrich Wansing, eds.), King's College Publications, vol. 5, 2005, pp. 309 – 333.

[Str02] Lutz Straßburger, *A local system for linear logic*, Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2002 (Matthias Baaz and Andrei Voronkov, eds.), Lecture Notes in Artificial Intelligence, vol. 2514, Springer-Verlag, 2002, pp. 388–402.

[Str03a] Lutz Straßburger, *Linear logic and noncommutativity in the calculus of structures*, Ph.D. thesis, Technische Universität Dresden, 2003.

[Str03b] Lutz Straßburger, *MELL in the calculus of structures*, Theoretical Computer Science **309** (2003), 213–285.

[Str03c] ———, *System NEL is undecidable*, 10th Workshop on Logic, Language, Information and Computation (WoLLIC) (Ruy De Queiroz, Elaine Pimentel, and Lucília Figueiredo, eds.), Electronic Notes in Theoretical Computer Science, vol. 84, Elsevier, 2003.

[SU99] Morten Heine B. Sørensen Sørensen and Pawel Urzyczyn, *Lecture notes on the curry-howard isomorphism*, Lecture Notes of the 11th European Summer School in Logic, Language and Information (ESSLLI 99), 1999.

[Thi94] Michael Thielscher, *Representing Actions in Equational Logic Programming*, Proceedings of the International Conference on Logic Programming (ICLP) (Santa Margherita Ligure, Italy) (P. Van Hentenryck, ed.), MIT Press, 1994, pp. 207–224.

[Thi99] ———, *From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem*, Artificial Intelligence **(1–2) 111** (1999), 277–299.

[Thi01] ———, *The concurrent, continuous Fluent Calculus*, Studia Logica **67** (2001), no. 3, 315–331.

[Thi05] ———, *FLUX: A logic programming method for reasoning agents*, Theory and Practice of Logic Programming **5** (2005), no. 4–5, 533–565.

[Tiu01] Alwen Fernanto Tiu, *Properties of a logical system in the calculus of structures*, Tech. Report WV-01-06, Technische Universität Dresden, 2001, replaced by [Tiu05].

[Tiu06a] Alwen Tiu, *A local system for intuitionistic logic*, LPAR 2006 (M. Hermann and A. Voronkov, eds.), Lecture Notes in Artificial Intelligence, vol. 4246, Springer-Verlag, 2006, pp. 242–256.

[Tiu06b] ———, *A system of interaction and structure II: The need for deep inference*, Logical Methods in Computer Science **2** (2006), no. 2, 4:1–24.

[TS96] Anne Sjerp Troelstra and Helmut Schwichtenberg, *Basic proof theory*, Cambridge University Press, 1996.

[vdBMV03] Mark van den Brand, Pierre-Etienne Moreau, and Jurgen Vinju, *A generator of efficient strongly typed abstract syntax trees in java*, Tech. Report SEN-E0306, ISSN 1386-369X, CWI, Amsterdam (Holland), November 2003.

[VMO03] Alberto Verdejo and Narciso Martí-Oliet, *Executable structural operational semantics in Maude*, Tech. Report 134-03, Dpto. Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2003, Available on the web at http://www.ucm.es/sip/alberto.

[Wel94] D. S. Weld, *An introduction to least commitment planning*, AI Magazine **15(4)** (1994), 27–61.

[Wel99] ———, *Recent advances in AI planning*, AI Magazine **20(2)** (1999), 93–122.

[WN95] Glynn Winskel and Morgens Nielsen, *Models for concurrency*, Handbook of Logic in Computer Science, vol. 4, Oxford University Press, 1995, pp. 1–148.

[Yet90] David N. Yetter, *Quantales and (non-commutative) linear logic*, Journal of Symbolic Logic **55 (1)** (1990), 41–64.

Index